Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-96-06

Case Studies of Non-Freely Generated Data Types

Claus Sengler

October 1996

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Abstract

In this report we shall present case studies of different data type specifications for natural numbers, for integers (int and int2), for finite lists (list and list2), for finite lists with an additional error element, for finite sets (set and set2), for binary words, for commutative trees, and for arrays. Furthermore, this report contains a collection of constructive function and predicate specifications, whose recursion orderings are shown to be well-founded.

Contents

1. Introduction **1**

2. Natural Numbers, nat **3**

2.1 $+ : \mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{nat}$ 4

2.2 $- : \mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{nat}$ 4

2.3 $<_{\mathsf{nat}}: \mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{bool}$ 7

2.4 $\leq_{\mathsf{nat}}$ $\mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{bool}$ 12

2.5 $>_{\mathsf{nat}}$ $\mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{bool}$ 12

2.6 $\geq_{\mathsf{nat}}$ $\mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{bool}$ 12

3. Integers, int **13**

4. Integers, int2 **14**

4.1 $\mathsf{succ} : \mathsf{int2} \rightarrow \mathsf{int2}$ 15

4.2 $\mathsf{pred} : \mathsf{int2} \rightarrow \mathsf{int2}$ 15

4.3 $+ : \mathsf{int2} \times \mathsf{int2} \rightarrow \mathsf{int2}$ 15

4.4 $\mathsf{negate} : \mathsf{int2} \rightarrow \mathsf{int2}$ 16

4.5 $- : \mathsf{int2} \times \mathsf{int2} \rightarrow \mathsf{int2}$ 16

4.6 $<_{\mathsf{int2}}: \mathsf{int2} \times \mathsf{int2} \rightarrow \mathsf{bool}$ 16

4.7 $\leq_{\mathsf{int2}}: \mathsf{int2} \times \mathsf{int2} \rightarrow \mathsf{bool}$ 17

4.8 $>_{\mathsf{int2}}: \mathsf{int2} \times \mathsf{int2} \rightarrow \mathsf{bool}$ 17

4.9 $\geq_{\mathsf{int2}}: \mathsf{int2} \times \mathsf{int2} \rightarrow \mathsf{bool}$ 17

5. Finite Lists, list **18**
5.1 app : list × list → list 19
5.2 member : nat × list → bool 19
5.3 length : list → nat 20
5.4 delete : nat × list → list 21
5.5 min : list → nat 23
5.6 max : list → nat 25
5.7 last : list → nat 27
5.8 butlast : list → list 28
5.9 sort : list → list 30
5.10 $<_{\mathsf{list}}$: list × list → bool 31
5.11 $\leq_{\mathsf{list}}$: list × list → bool 36
5.12 $>_{\mathsf{list}}$: list × list → bool 36
5.13 $\geq_{\mathsf{list}}$: list × list → bool 36

6. Finite Lists, list2 **37**
6.1 cons : nat × list2 → list2 44
6.2 member : nat × list2 → bool 44
6.3 length : list2 → nat 45
6.4 delete : nat × list2 → list2 47
6.5 last : list2 → nat 52
6.6 butlast : list2 → list2 53
6.7 $<_{\mathsf{list2}}$: list2 × list2 → bool 56
6.8 $\leq_{\mathsf{list2}}$: list2 × list2 → bool 56
6.9 $>_{\mathsf{list2}}$: list2 × list2 → bool 56
6.10 $\geq_{\mathsf{list2}}$: list2 × list2 → bool 56

7. Error Lists, errorlist **57**
7.1 app : errorlist × errorlist → errorlist 62
7.2 member : nat × errorlist → bool 63
7.3 length : errorlist → nat 64
7.4 delete : nat × errorlist → errorlist 65
7.5 min : errorlist → nat 67
7.6 max : errorlist → nat 69
7.7 last : errorlist → nat 70
7.8 butlast : errorlist → errorlist 71
7.9 sort : errorlist → errorlist 74
7.10 $<_{\mathsf{errorlist}}$: errorlist × errorlist → bool 75
7.11 $\leq_{\mathsf{errorlist}}$: errorlist × errorlist → bool 82
7.12 $>_{\mathsf{errorlist}}$: errorlist × errorlist → bool 82
7.13 $\geq_{\mathsf{errorlist}}$: errorlist × errorlist → bool 82

8. Finite Sets, set **83**
8.1 delete : nat × set → set 88
8.2 union : set × set → set 90
8.3 inter : set × set → set 91
8.4 diff : set × set → set 94

8.5 min : set → nat 98
8.6 max : set → nat 99
8.7 card : set → nat 101
8.8 sort : set → list 102
8.9 $<_{\mathsf{set}}$: set × set → bool 103
8.10 $\leq_{\mathsf{set}}$: set × set → bool 108
8.11 $>_{\mathsf{set}}$: set × set → bool 108
8.12 $\geq_{\mathsf{set}}$: set × set → bool 108

9. Finite Sets, set2 **109**
9.1 delete : nat × set2 → set2 120
9.2 ins : nat × set2 → set2 124
9.3 inter : set2 × set2 → set2 125
9.4 diff : set2 × set2 → set2 129
9.5 card : set2 → nat 133
9.6 $<_{\mathsf{set2}}$: set2 × set2 → bool 135
9.7 $\leq_{\mathsf{set2}}$: set2 × set2 → bool 135
9.8 $>_{\mathsf{set2}}$: set2 × set2 → bool 135
9.9 $\geq_{\mathsf{set2}}$: set2 × set2 → bool 135

10. Binary Words, binword **136**
10.1 succ : binword → binword 145
10.2 pred : binword → binword 145
10.3 + : binword × binword → binword 146
10.4 − : binword × binword → binword 147
10.5 $<_{\mathsf{binword}}$: binword × binword → bool 148
10.6 $\leq_{\mathsf{binword}}$: binword × binword → bool 148
10.7 $>_{\mathsf{binword}}$: binword × binword → bool 149
10.8 $\geq_{\mathsf{binword}}$: binword × binword → bool 149

11. Commutative Trees, tree **150**
11.1 count : tree → nat 155
11.2 height : tree → nat 156
11.3 leafcount : tree → nat 158
11.4 delete : nat × tree → tree 159
11.5 $<_{\mathsf{tree}}$: tree × tree → bool 163
11.6 $\leq_{\mathsf{tree}}$: tree × tree → bool 163
11.7 $>_{\mathsf{tree}}$: tree × tree → bool 163
11.8 $\geq_{\mathsf{tree}}$: tree × tree → bool 163

12. Arrays, array **164**
12.1 delete : nat × array → array 169
12.2 size : array → nat 171
12.3 min_index : array → nat 172
12.4 index_min : array → nat 174
12.5 swap : nat × nat × array → nat 176
12.6 sort : array → nat 182

12.7 $<_{\mathsf{array}}$: $\mathsf{array} \times \mathsf{array} \rightarrow \mathsf{bool}$ 183
12.8 $\leq_{\mathsf{array}}$: $\mathsf{array} \times \mathsf{array} \rightarrow \mathsf{bool}$ 183
12.9 $>_{\mathsf{array}}$: $\mathsf{array} \times \mathsf{array} \rightarrow \mathsf{bool}$ 183
12.10 $\geq_{\mathsf{array}}$: $\mathsf{array} \times \mathsf{array} \rightarrow \mathsf{bool}$ 183

1 Introduction

The induction principle is based on well-founded orderings, i.e., orderings without infinite descending chains. Therefore, in order to automate inductive proofs, it is essential to automate the proofs for an ordering to be well-founded, which is closely related to a termination proof.

The general idea to achieve these proofs is to use the $\prec_{\mathbb{N}}$-relation on natural numbers and to map each data type object into a natural number by use of a measure function. For an automation typically a single measure function is used, for instance, the size of an object.

In case of freely generated data types, that is for data types whose objects possess a unique syntactic structure, like, for instance, natural numbers, finite lists, and finite trees, the size of an object corresponds to the number of reflexive constructor functions that are necessary to represent the object. Here, the axiomatization of a function to compute the size of an object can be easily encoded in a first-order logic. Thus, together with an axiomatization of the natural numbers the occurring proof obligations can be proved quite easily.

Besides freely generated data types there are non-freely generated data types which frequently occur in practical applications. These data types include, for example, finite sets and arrays. They are characterized by having objects with different syntactic representations. The size of such an object corresponds to the *minimal* number of reflexive constructor functions that are necessary to represent the object. Compared to freely generated data types, the size of a non-freely generated data type object can only be axiomatized within a first-order logic in a complicated and inconstructive way, which leads to substantially more difficult proofs.

Yet, for the proof obligations that occur during the proofs of an ordering to be well-founded it is not necessary to compute the size of an object explicitly. Instead it is sufficient to estimate how two objects relate to each other with respect to their size. This idea is incorporated into a specific calculus, the Estimation Calculus, which was originally designed by Walther for termination proofs over freely generated data types.

In DFKI Technical Report RR-96-01, "Induction on Non-Freely Generated Data Types", we present a generalization of this calculus that allows to efficiently derive estimations for non-freely generated data type objects with respect to their size, too.

For the resulting proof obligations when proving an ordering to be well-founded, the Estimation Calculus decides whether under a condition φ the sizes of two objects, which are denoted by terms, are within the $\prec_{\mathbb{N}}$-relation. Moreover, usually both terms possess a common subterm. Hence, the proof obligations are of the form:

$$\varphi \rightarrow |\mathsf{f(t)}| \prec_{\mathbb{N}} |\mathsf{g(t)}|.$$

In order to prove this obligation, it is split by the Estimation Calculus into a chain of estimations with respect to the $\preceq_{\mathbb{N}}$-relation. Furthermore, each single estimation is based on certain properties of the underlying formal specification:

$$\varphi \rightarrow |\mathsf{f(t)}| \preceq_{\mathbb{N}} |\mathsf{t}| \text{ and}$$

$$\varphi \rightarrow |\mathsf{t}| \preceq_{\mathbb{N}} |\mathsf{g(t)}|.$$

These single estimations together with the transitivity of the $\preceq_{\mathbb{N}}$-Relation guarantee

$$\varphi \rightarrow |\mathsf{f(t)}| \preceq_{\mathbb{N}} |\mathsf{g(t)}|.$$

To show that the objects which are denoted by the two terms are within the strict $\prec_{\mathbb{N}}$-relation, for each single estimation a formula is synthesized which is sufficient for the $\prec_{\mathbb{N}}$-relation. Then, an additional proof that the disjunction of these formulas follows from the condition φ guarantees the original proof obligation.

For the single estimations within the Estimation Calculus certain properties of the involved functions are used. Defined functions are analyzed whether they are argument-bounded, i.e., whether the size of one of their arguments denotes an upper bound for the size of an application of the function. And for constructor functions it is determined, whether the size of each argument is a lower bound for the size of an application of the function.

Whereas the first property can be proved within the Estimation Calculus itself, the second property is more difficult to show. To do that an implementation of the non-freely generated data type as a freely generated data type has to be used. This allows one to axiomatize a function that computes the size of an object explicitly, however, in general, in an inconstructive way. Together with an axiomatization of the natural numbers, the second property can be encoded in a first-order logic, and, thus, be proved.

Hence, for a data type specification certain properties of the involved functions are proved in advance, in order to allow the use of these properties later on for proofs of orderings being well-founded. Thereby, this approach together with the generalized Estimation Calculus enables an efficient automation of the proofs for an ordering to be well-founded.

In this report we shall present a collection of different data type specifications for natural numbers, for integers (int and int2), for finite lists (list and list2), for finite lists with an additional error element, for finite sets (set and set2), for binary words, for commutative trees, and for arrays. For all of these data type specifications it is determined according to our approach described in DFKI Technical Report RR-96-01, whether their reflexive constructor functions are size increasing, in which case the respective strictness and minimal representation predicates are specified. Furthermore, this report contains a collection of constructive function and predicate specifications, whose recursion orderings are shown to be well-founded.

2

Natural Numbers, nat

Our specification of natural numbers, nat uses two constructor functions $\mathsf{0} :\to \mathsf{nat}$, generating zero, and $\mathsf{succ} : \mathsf{nat} \to \mathsf{nat}$, generating the successor of a number. Equality on nats is specified by the axioms:

$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{0} \not\equiv \mathsf{succ}(\mathsf{x})$ and

$\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{nat}\ \mathsf{succ}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{y}) \to \mathsf{x} \equiv \mathsf{y}$.

By the above specification we have defined a freely generated data type. Hence, the constructor function succ is size increasing, and we can synthesize the strictness predicate $\Theta^1_{\mathsf{succ}} : \mathsf{nat} \to \mathsf{bool}$ and the minimal representation predicate $\Gamma_{\mathsf{succ}} : \mathsf{nat} \to \mathsf{bool}$ by

$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \Theta^1_{\mathsf{succ}}(\mathsf{x}) \equiv \mathsf{true}$ and

$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \Gamma_{\mathsf{succ}}(\mathsf{x}) \equiv \mathsf{true}$.

Furthermore, the constructor functions of nat are non-overlapping, which leads to the following synthesis of the destructor function $\mathsf{pred} : \mathsf{nat} \to \mathsf{nat}$ for the constructor function succ:

$\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{nat}\ \mathsf{x} \equiv \mathsf{succ}(\mathsf{y}) \to \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))$,

$\mathsf{pred}(\mathsf{0}) \equiv \mathsf{0}$, and

$\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{nat}\ \mathsf{x} \equiv \mathsf{succ}(\mathsf{y}) \to \Gamma_{\mathsf{succ}}(\mathsf{pred}(\mathsf{x})) \equiv \mathsf{true}$.

Furthermore, pred is 1-bounded with difference predicate $\Delta^1_{\mathsf{pred}} : \mathsf{nat} \to \mathsf{bool}$:

$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))$

For the data type nat we will give constructive function and predicate specifications for $+$, $-$, $<_{\mathsf{nat}}$, $\leq_{\mathsf{nat}}$, $>_{\mathsf{nat}}$, and $\geq_{\mathsf{nat}}$.

2.1 $+ : \mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$

$+$ computes the addition on natural numbers and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad \mathsf{x} \equiv 0 \to (\mathsf{x} + \mathsf{y}) \equiv \mathsf{y} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \to (\mathsf{x} + \mathsf{y}) \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}) + \mathsf{y}) \end{array}$$

The recursion ordering of $+$ is well-founded: There is only one definition case with a single recursive call of $+$. Hence, we use the Estimation Calculus, abbreviating the invariant case condition

$$\mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))$$

by φ:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{pred}(\mathsf{x}) \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \lor \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true} \right\rangle}\ \text{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{nat}}$-relation, we have to show

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat} \\ \quad \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \\ \qquad \to (\mathsf{false} \lor \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true}), \end{array}$$

which can be simplified using the definition of Δ^1_{pred} to

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat} \\ \quad \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \to \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})). \end{array}$$

2.2 $- : \mathsf{nat} \times \mathsf{nat} \to \mathsf{nat}$

$-$ computes the subtraction on natural numbers and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv 0) \to (\mathsf{x} - \mathsf{y}) \equiv \mathsf{x} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \land \mathsf{x} \equiv 0) \to (\mathsf{x} - \mathsf{y}) \equiv 0 \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \land \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))) \\ \qquad \to (\mathsf{x} - \mathsf{y}) \equiv (\mathsf{pred}(\mathsf{x}) - \mathsf{pred}(\mathsf{y})) \end{array}$$

The recursion ordering of $-$ is well-founded: There is only one definition case with a single recursive call of $-$. For each argument we use the Estimation Calculus, abbreviating the invariant case condition

$$(\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})))$$

by φ. For the first argument of $-$, x, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{pred}(\mathsf{x}) \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{nat}}$-relation, we have to show

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true}), \end{array}$$

which can be simplified using the definition of Δ^1_{pred} to

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))) \\ \qquad \to \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})). \end{array}$$

And for the second argument of $-$t, y, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{y} \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{pred}(\mathsf{y}) \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{y}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{nat}}$-relation, we have to show

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{y}) \equiv \mathsf{true}), \end{array}$$

which can be simplified using the definition of Δ^1_{pred} to

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))) \\ \qquad \to \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})). \end{array}$$

Hence, the recursion ordering of $-$ is a well-founded order relation.

In addition, $-$ denotes a 1-bounded function symbol. To prove this property, first of all, we need to show that $-$ is completely specified, by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv 0) \vee \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv 0) \vee \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))) \end{array}$$

Then, we examine each definition case separately. For the first case we obtain

$$\begin{array}{c} - \\ \text{——— Identity ———} \\ \langle \mathsf{y} \equiv 0, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \rangle \end{array}$$

For the second case we abbreviate the invariant case condition

$$\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \land \mathsf{x} \equiv 0$$

by φ. Using the Estimation Calculus, we obtain

$$\begin{array}{c} - \\ \text{—— Identity ——} \\ \langle \varphi, 0 \preceq_{\mathsf{nat}} 0, \mathsf{false} \rangle \\ \text{— Equation 1 —} \\ \langle \varphi, 0 \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \rangle \end{array}$$

For the third case we abbreviate the invariant case condition

$$\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \land \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))$$

by φ. Furthermore, since this is a recursive case, we may use the additional inference rule

$$\xi \Rightarrow \quad \begin{array}{c} \langle \varphi, \mathsf{pred}(\mathsf{x}) \preceq_{\mathsf{nat}} \mathsf{x}, \Delta \rangle \\ \text{—————— Induction Hypothesis ——————} \\ \langle \varphi, (\mathsf{pred}(\mathsf{x}) - \mathsf{pred}(\mathsf{y})) \preceq_{\mathsf{nat}} \mathsf{pred}(\mathsf{x}), \Delta^1_-(\mathsf{pred}(\mathsf{x}), \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \rangle \end{array}$$

where ξ is an abbreviation for the formula

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \varphi \rightarrow \Delta$$

as an induction hypothesis. Now, the derivation in the Estimation Calculus is given by

$$\begin{array}{c} - \\ \text{——————— Identity ———————} \\ \langle \varphi, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \rangle \\ \text{——————— Estimation ———————} \\ \left\langle \varphi, \mathsf{pred}(\mathsf{x}) \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \lor \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true} \right\rangle \\ \text{————— Induction Hypothesis —————} \\ \left\langle \begin{array}{c} \varphi, (\mathsf{pred}(\mathsf{x}) - \mathsf{pred}(\mathsf{y})) \preceq_{\mathsf{nat}} \mathsf{pred}(\mathsf{x}), \\ \Delta^1_-(\mathsf{pred}(\mathsf{x}), \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \end{array} \right\rangle \\ \text{————— Strong Embedding —————} \\ \left\langle \begin{array}{c} \varphi, (\mathsf{pred}(\mathsf{x}) - \mathsf{pred}(\mathsf{y})) \preceq_{\mathsf{nat}} \mathsf{succ}(\mathsf{pred}(\mathsf{x})), \\ \Delta^1_-(\mathsf{pred}(\mathsf{x}), \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \lor \Theta^1_{\mathsf{succ}}(\mathsf{pred}(\mathsf{x})) \equiv \mathsf{true} \end{array} \right\rangle \\ \text{——————— Equation 4 ———————} \\ \left\langle \begin{array}{c} \varphi, (\mathsf{pred}(\mathsf{x}) - \mathsf{pred}(\mathsf{y})) \preceq_{\mathsf{nat}} \mathsf{x}, \\ \Delta^1_-(\mathsf{pred}(\mathsf{x}), \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \lor \Theta^1_{\mathsf{succ}}(\mathsf{pred}(\mathsf{x})) \equiv \mathsf{true} \end{array} \right\rangle \end{array}$$

where to apply the induction hypothesis, the formula

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \varphi \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true})$$

has to be proved.

Based on the different derivations in the Estimation Calculus and using the simplified difference formulas, the difference predicate for $-$, Δ^1_- : $\mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{bool}$ is synthesized as:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv 0) \rightarrow \Delta^1_-(\mathsf{x}, \mathsf{y}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv 0) \rightarrow \Delta^1_-(\mathsf{x}, \mathsf{y}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))) \\ \qquad \rightarrow \Delta^1_-(\mathsf{x}, \mathsf{y}) \equiv \mathsf{true} \end{array}$$

An additional simplification of this definition yields:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad \Delta^1_-(\mathsf{x}, \mathsf{y}) \equiv \mathsf{true} \leftrightarrow (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x}))). \end{array}$$

2.3 $<_{\mathsf{nat}}$: $\mathsf{nat} \times \mathsf{nat} \rightarrow \mathsf{bool}$

$<_{\mathsf{nat}}$ computes the less-than-relation on natural numbers and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv 0) \rightarrow (\mathsf{x} <_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv 0) \rightarrow (\mathsf{x} <_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}) \\ \qquad \rightarrow (\mathsf{x} <_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false}) \\ \qquad \rightarrow (\mathsf{x} <_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{false} \end{array}$$

The recursion ordering of $<_{\mathsf{nat}}$ is well-founded: There are two definition cases with a single recursive call of $<_{\mathsf{nat}}$ in each. For each recursive definition case and each argument we use the Estimation Calculus. Starting with the first recursive case, we abbreviate the invariant case condition

$$(\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true})$$

by φ. For the first argument of $<_{\mathsf{nat}}$, x, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{pred}(\mathsf{x}) \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true} \right\rangle}\ \text{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{nat}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}) \\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of Δ^1_{pred} to

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}) \\ \qquad \rightarrow \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})).\end{array}$$

And for the second argument of $<_{\mathsf{nat}}$, y, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{y} \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{pred}(\mathsf{y}) \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{y}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{nat}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}) \\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{y}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of Δ^1_{pred} to

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}) \\ \qquad \rightarrow \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})).\end{array}$$

For the second recursive definition case we abbreviate the invariant case condition

$$(\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false})$$

by φ. For the first argument of $<_{\mathsf{nat}}$, x, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{pred}(\mathsf{x}) \preceq_{\mathsf{nat}} \mathsf{x}, \mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{nat}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false}) \\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{x}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of Δ^1_{pred} to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false})\\ \qquad \to \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})).\end{array}$$

And for the second argument of $<_{\mathsf{nat}}$, y, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{y} \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{pred}(\mathsf{y}) \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{y}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{nat}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y}{:}\,\mathsf{nat}\\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false})\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred}}(\mathsf{y}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of Δ^1_{pred} to

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y}{:}\,\mathsf{nat}\\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false})\\ \qquad \to \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})).\end{array}$$

Thus, the recursion ordering of $<_{\mathsf{nat}}$ is a well-founded ordering.

In addition, $<_{\mathsf{nat}}$ denotes a well-founded ordering as well. To prove that, we first have to show that $<_{\mathsf{nat}}$ is completely specified, i.e.,

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y}{:}\,\mathsf{nat}\\ \quad (\mathsf{y} \equiv 0) \vee\\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv 0) \vee\\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true})) \vee\\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false}))\end{array}$$

Next, for each definition case we show that

$$\forall\, \mathsf{x}, \mathsf{y}{:}\,\mathsf{nat}\ (\mathsf{x} <_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{true} \to \mathsf{x} \prec_{\mathsf{nat}} \mathsf{y},$$

again, using the Estimation Calculus. For the first case we obtain

$$\dfrac{-}{\langle \mathsf{y} \equiv 0 \wedge \mathsf{false} \equiv \mathsf{true}, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{y}, \Delta_1\rangle}\ \text{Tautology}$$

where in order to enable the application of the Tautology Rule, the first-order formula

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y}{:}\,\mathsf{nat}\\ \quad \neg(\mathsf{y} \equiv 0 \wedge \mathsf{false} \equiv \mathsf{true})\end{array}$$

has to be proved. To prove the strict relation, the formula

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\\ \quad (\mathsf{y} \equiv 0 \wedge \mathsf{false} \equiv \mathsf{true}) \rightarrow \Delta_1\end{array}$$

has to be shown. For the second case we obtain the derivation

$$\dfrac{\dfrac{\dfrac{-}{\langle \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv 0 \wedge \mathsf{true} \equiv \mathsf{true}, 0 \preceq_{\mathsf{nat}} \mathsf{succ}(\mathsf{pred}(\mathsf{y})), \mathsf{true}\rangle}\ \text{Strong Estimation}}{\langle \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv 0 \wedge \mathsf{true} \equiv \mathsf{true}, 0 \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{true}\rangle}\ \text{Equation 1}}{\langle \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv 0 \wedge \mathsf{true} \equiv \mathsf{true}, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{true}\rangle}\ \text{Equation 5}$$

showing the strict relation by

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\\ \quad (\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv 0 \wedge \mathsf{true} \equiv \mathsf{true})\\ \qquad \rightarrow \mathsf{true}\end{array}$$

The third definition case is a recursive case. Hence, we need to make an additional case analysis:

$(\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}$ or

$(\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false}.$

For the first case we can assume as an induction hypothesis the inference rule:

$$\dfrac{-}{\langle \varphi, \mathsf{pred}(\mathsf{x}) \preceq_{\mathsf{nat}} \mathsf{pred}(\mathsf{y}), \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where we use φ as an abbreviation for

$$\left(\begin{array}{c}\mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge\\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}\end{array}\right)$$

Then, the derivation of

$\langle \varphi, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{y}, \Delta_3\rangle$

is achieved by:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{pred}(\mathsf{x}) \preceq_{\mathsf{nat}} \mathsf{pred}(\mathsf{y}), \mathsf{true}\rangle}\ \text{Induction Hypothesis}}{\langle \varphi, \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \preceq_{\mathsf{nat}} \mathsf{succ}(\mathsf{pred}(\mathsf{y})), \mathsf{true} \vee \Gamma_{\mathsf{succ}}(\mathsf{pred}(\mathsf{x})) \equiv \mathsf{false}\rangle}\ \text{Weak Embedding}}{\langle \varphi, \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{true} \vee \Gamma_{\mathsf{succ}}(\mathsf{pred}(\mathsf{x})) \equiv \mathsf{false}\rangle}\ \text{Equation 3}}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{y}, \mathsf{true} \vee \Gamma_{\mathsf{succ}}(\mathsf{pred}(\mathsf{x})) \equiv \mathsf{false}\rangle}\ \text{Equation 5}$$

where in order to enable the application of the Weak Embedding Rule, the first-order formula

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \varphi \to \Gamma_{\mathsf{succ}}(\mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}$$

has to be shown. The strict relation is proved by

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\\ \quad \varphi \to (\mathsf{true} \vee \Gamma_{\mathsf{succ}}(\mathsf{pred}(\mathsf{x})) \equiv \mathsf{false}).\end{array}$$

For the second case we cannot assume an induction hypothesis. We prove the estimation formula

$$\left\langle \begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false}, \\ \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{y}, \Delta_4 \end{array} \right\rangle$$

by an application of the Tautology Rule, where in order to enable this application, it is necessary to prove the first-order formula:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\\ \quad \neg \left(\begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \end{array} \right)\end{array}$$

To prove the strict relation, the formula

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\\ \quad \left(\begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \end{array} \right)\\ \quad \to \Delta_4\end{array}$$

needs to be shown.

The fourth definition case is also a recursive case. Hence, we need to make an additional case analysis:

$(\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}$ or

$(\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false}$.

Although for the first case we could assume an induction hypothesis, this is not necessary since the derivation of the estimation formula

$$\left\langle \begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true}, \\ \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{y}, \Delta_5 \end{array} \right\rangle$$

can be achieved by the application of the Tautology Rule. In order to enable this application, the first-order formula

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\\ \quad \neg \left(\begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \end{array} \right)\end{array}$$

has to be shown. And for the strict relation, the formula

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad \left(\begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{true} \end{array}\right) \\ \quad \to \Delta_5 \end{array}$$

needs to be proved. For the second case we cannot assume an induction hypothesis. We prove the estimation formula

$$\left\langle \begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false}, \\ \mathsf{x} \preceq_{\mathsf{nat}} \mathsf{y}, \Delta_6 \end{array} \right\rangle$$

by an application of the Tautology Rule, where in order to enable this application, it is necessary to prove the first-order formula:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad \neg \left(\begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \end{array}\right) \end{array}$$

To prove the strict relation, the formula

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad \left(\begin{array}{c} \mathsf{y} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{y})) \wedge \mathsf{x} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \wedge \\ (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{pred}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{pred}(\mathsf{y})) \equiv \mathsf{false} \end{array}\right) \\ \quad \to \Delta_6 \end{array}$$

needs to be shown.

Having proved all these obligations, $<_{\mathsf{nat}}$ denotes a well-founded order relation.

2.4 $\leq_{\mathsf{nat}}$ $\mathsf{nat} \times \mathsf{nat} \to \mathsf{bool}$

$\leq_{\mathsf{nat}}$ computes the less-than-or-equal-relation on natural numbers and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{x} \leq_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow ((\mathsf{x} <_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{true} \vee \mathsf{x} \equiv \mathsf{y}) \end{array}$$

Since this a non-recursive constructive specification, we are done.

2.5 $>_{\mathsf{nat}}$ $\mathsf{nat} \times \mathsf{nat} \to \mathsf{bool}$

$>_{\mathsf{nat}}$ computes the greater-than-relation on natural numbers and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{x} >_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow (\mathsf{y} <_{\mathsf{nat}} \mathsf{x}) \equiv \mathsf{true} \end{array}$$

Since this a non-recursive constructive specification, we are done.

2.6 $\geq_{\mathsf{nat}}$ $\mathsf{nat} \times \mathsf{nat} \to \mathsf{bool}$

$\geq_{\mathsf{nat}}$ computes the greater-than-or-equal-relation on natural numbers and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad (\mathsf{x} \geq_{\mathsf{nat}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow (\mathsf{y} \leq_{\mathsf{nat}} \mathsf{x}) \equiv \mathsf{true} \end{array}$$

Since this a non-recursive constructive specification, we are done.

3

Integers, int

This specification of integers, int, uses three constructor functions $\mathsf{0} :\to \mathsf{int}$, generating zero, $\mathsf{succ} : \mathsf{int} \to \mathsf{int}$, generating the successor of an integer, and $\mathsf{pred} : \mathsf{int} \to \mathsf{int}$, generating the predecessor of an integer. Equality on int is specified by the axioms:

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{int} \quad \mathsf{succ}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{y}) \to \mathsf{x} \equiv \mathsf{y}$$

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{int} \quad \mathsf{pred}(\mathsf{x}) \equiv \mathsf{pred}(\mathsf{y}) \to \mathsf{x} \equiv \mathsf{y}$$

$$\forall\, \mathsf{x} : \mathsf{int} \quad \mathsf{succ}(\mathsf{pred}(\mathsf{x})) \equiv \mathsf{x}$$

$$\forall\, \mathsf{x} : \mathsf{int} \quad \mathsf{pred}(\mathsf{succ}(\mathsf{x})) \equiv \mathsf{x}$$

$$\forall\, \mathsf{x} : \mathsf{int} \quad \mathsf{pred}(\mathsf{x}) \not\equiv \mathsf{succ}(\mathsf{x})$$

By the above specification we have defined a non-freely generated data type, since, for example, $\mathsf{0} \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{0}))$. The minimal size of a data type object, i.e., an integer, corresponds to the absolute value of the number. Unfortunately, both constructor functions succ and pred are *not* size increasing. Hence, we cannot use the Estimation Calculus to prove orderings to be well-founded.

4

Integers, int2

This specification of integers, int2, uses a single constructor function $\mathsf{make\_int} : \mathsf{sign} \times \mathsf{nat} \to \mathsf{int2}$, generating an integer as a signed natural number. Thereby, the data type sign is specified as a freely generated data type with two constructor functions $+ :\to \mathsf{sign}$ and $- :\to \mathsf{sign}$. Equality on int2 is specified by the axiom:

$$\begin{array}{l} \forall\, \mathsf{i}, \mathsf{j} : \mathsf{sign}\ \forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \\ \quad \mathsf{make\_int}(\mathsf{i}, \mathsf{x}) \equiv \mathsf{make\_int}(\mathsf{j}, \mathsf{y}) \\ \qquad \leftrightarrow \left(\begin{array}{c} (\mathsf{x} \equiv 0 \wedge \mathsf{y} \equiv 0) \vee \\ (\mathsf{i} \equiv \mathsf{j} \wedge \mathsf{x} \equiv \mathsf{y}) \end{array} \right) \end{array}$$

By the above specification we have defined a non-freely generated data type, since

$$\mathsf{make\_int}(+, 0) \equiv \mathsf{make\_int}(-, 0).$$

Since the single constructor function make_int is irreflexive, there is no need to prove a constructor function to be size increasing.

As example algorithms we will only use non-recursive function and predicate specifications. Otherwise, we would have to refine our general approach by using a different measure function. For the structural ordering of the data type int2 this means to compare two integers by the sizes of their absolute values.

Still, we have to define the destructor functions for the constructor function make_int, $\mathsf{sign} : \mathsf{int2} \to \mathsf{sign}$ for the first argument of make_int, and $\mathsf{abs} : \mathsf{int2} \to \mathsf{nat}$ for the second argument of make_int. Do not be confused that we use the symbol sign for both, a sort name and a function name. Then, sign and abs are defined by:

$$\begin{array}{l} \forall\, \mathsf{i}:\mathsf{sign}\ \forall\, \mathsf{y}:\mathsf{nat}\ \forall\, \mathsf{x}:\mathsf{int2} \\ \quad \mathsf{x} \equiv \mathsf{make\_int}(\mathsf{i},\mathsf{y}) \rightarrow \mathsf{x} \equiv \mathsf{make\_int}(\mathsf{sign}(\mathsf{x}),\mathsf{abs}(\mathsf{x})) \end{array}$$

For the data type int2 we will now give constructive function and predicate specifications for succ, pred, $+$, negate, $-$, $<_{\mathsf{int2}}$, $\leq_{\mathsf{int2}}$, $>_{\mathsf{int2}}$, and $\geq_{\mathsf{int2}}$.

4.1 succ : int2 → int2

succ computes the successor of an integer and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}:\mathsf{int2} \\ \quad \mathsf{abs}(\mathsf{x}) \equiv 0 \rightarrow \mathsf{succ}(\mathsf{x}) \equiv \mathsf{make\_int}(+,\mathsf{succ}(0)) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}:\mathsf{int2} \\ \quad (\mathsf{abs}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{abs}(\mathsf{x}))) \wedge \mathsf{sign}(\mathsf{x}) \equiv +) \\ \qquad \rightarrow \mathsf{succ}(\mathsf{x}) \equiv \mathsf{make\_int}(+,\mathsf{succ}(\mathsf{abs}(\mathsf{x}))) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}:\mathsf{int2} \\ \quad (\mathsf{abs}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{abs}(\mathsf{x}))) \wedge \mathsf{sign}(\mathsf{x}) \equiv -) \\ \qquad \rightarrow \mathsf{succ}(\mathsf{x}) \equiv \mathsf{make\_int}(-,\mathsf{pred}(\mathsf{abs}(\mathsf{x}))) \end{array}$$

Note, in the above specification we use two different functions succ, one on integers and one on natural numbers. However, from the context it is clear which one is meant. In subsequent sections we will do so for other functions as well.

Since the above specification is non-recursive, we are done.

4.2 pred : int2 → int2

pred computes the predecessor of an integer and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}:\mathsf{int2} \\ \quad \mathsf{abs}(\mathsf{x}) \equiv 0 \rightarrow \mathsf{pred}(\mathsf{x}) \equiv \mathsf{make\_int}(-,\mathsf{succ}(0)) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}:\mathsf{int2} \\ \quad (\mathsf{abs}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{abs}(\mathsf{x}))) \wedge \mathsf{sign}(\mathsf{x}) \equiv +) \\ \qquad \rightarrow \mathsf{pred}(\mathsf{x}) \equiv \mathsf{make\_int}(+,\mathsf{pred}(\mathsf{abs}(\mathsf{x}))) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}:\mathsf{int2} \\ \quad (\mathsf{abs}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{abs}(\mathsf{x}))) \wedge \mathsf{sign}(\mathsf{x}) \equiv -) \\ \qquad \rightarrow \mathsf{pred}(\mathsf{x}) \equiv \mathsf{make\_int}(-,\mathsf{succ}(\mathsf{abs}(\mathsf{x}))) \end{array}$$

Since this specification is non-recursive, we are done.

4.3 + : int2 × int2 → int2

$+$ computes the addition on integers and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x},\mathsf{y}:\mathsf{int2} \\ \quad (\mathsf{sign}(\mathsf{x}) \equiv + \wedge \mathsf{sign}(\mathsf{y}) \equiv +) \\ \qquad \rightarrow (\mathsf{x}+\mathsf{y}) \equiv \mathsf{make\_int}(+,(\mathsf{abs}(\mathsf{x})+\mathsf{abs}(\mathsf{y}))) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2} \\ \quad (\mathsf{sign}(\mathsf{x}) \equiv - \wedge \mathsf{sign}(\mathsf{y}) \equiv -) \\ \qquad \to (\mathsf{x} + \mathsf{y}) \equiv \mathsf{make\_int}(-, (\mathsf{abs}(\mathsf{x}) + \mathsf{abs}(\mathsf{y}))) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2} \\ \quad (\mathsf{sign}(\mathsf{x}) \equiv + \wedge \mathsf{sign}(\mathsf{y}) \equiv - \wedge (\mathsf{abs}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{abs}(\mathsf{y})) \equiv \mathsf{true}) \\ \qquad \to (\mathsf{x} + \mathsf{y}) \equiv \mathsf{make\_int}(-, (\mathsf{abs}(\mathsf{y}) - \mathsf{abs}(\mathsf{x}))) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2} \\ \quad (\mathsf{sign}(\mathsf{x}) \equiv + \wedge \mathsf{sign}(\mathsf{y}) \equiv - \wedge (\mathsf{abs}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{abs}(\mathsf{y})) \equiv \mathsf{false}) \\ \qquad \to (\mathsf{x} + \mathsf{y}) \equiv \mathsf{make\_int}(+, (\mathsf{abs}(\mathsf{x}) - \mathsf{abs}(\mathsf{y}))) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2} \\ \quad (\mathsf{sign}(\mathsf{x}) \equiv - \wedge \mathsf{sign}(\mathsf{y}) \equiv + \wedge (\mathsf{abs}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{abs}(\mathsf{y})) \equiv \mathsf{true}) \\ \qquad \to (\mathsf{x} + \mathsf{y}) \equiv \mathsf{make\_int}(+, (\mathsf{abs}(\mathsf{y}) - \mathsf{abs}(\mathsf{x}))) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2} \\ \quad (\mathsf{sign}(\mathsf{x}) \equiv - \wedge \mathsf{sign}(\mathsf{y}) \equiv + \wedge (\mathsf{abs}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{abs}(\mathsf{y})) \equiv \mathsf{false}) \\ \qquad \to (\mathsf{x} + \mathsf{y}) \equiv \mathsf{make\_int}(-, (\mathsf{abs}(\mathsf{x}) - \mathsf{abs}(\mathsf{y}))) \end{array}$$

Since this specification is non-recursive, we are done.

4.4 negate : int2 → int2

negate computes the negation of an integer and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{int2} \\ \quad \mathsf{abs}(\mathsf{x}) \equiv 0 \to \mathsf{negate}(\mathsf{x}) \equiv \mathsf{make\_int}(+, 0) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, : \mathsf{int2} \\ \quad (\mathsf{abs}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{abs}(\mathsf{x}))) \wedge \mathsf{sign}(\mathsf{x}) \equiv +) \\ \qquad \to \mathsf{negate}(\mathsf{x}) \equiv \mathsf{make\_int}(-, \mathsf{abs}(\mathsf{x})) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}, : \mathsf{int2} \\ \quad (\mathsf{abs}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{pred}(\mathsf{abs}(\mathsf{x}))) \wedge \mathsf{sign}(\mathsf{x}) \equiv -) \\ \qquad \to \mathsf{negate}(\mathsf{x}) \equiv \mathsf{make\_int}(+, \mathsf{abs}(\mathsf{x})) \end{array}$$

Since this specification is non-recursive, we are done.

4.5 − : int2 × int2 → int2

− computes the subtraction of two integers and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2} \\ \quad (\mathsf{x} - \mathsf{y}) \equiv (\mathsf{x} + \mathsf{negate}(\mathsf{y})) \end{array}$$

Since this specification is non-recursive, we are done.

4.6 $<_{\mathsf{int2}}$: int2 × int2 → bool

$<_{\mathsf{int2}}$ computes the less-than-relation on integers and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2}\\ \quad (\mathsf{x} <_{\mathsf{int2}} \mathsf{y}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \begin{pmatrix} \mathsf{sign}(\mathsf{y}-\mathsf{x}) \equiv + \wedge \\ \mathsf{abs}(\mathsf{y}-\mathsf{x}) \not\equiv 0 \end{pmatrix}\end{array}$$

Since this specification is non-recursive, we are done.

4.7 $\leq_{\mathsf{int2}}$: int2 × int2 → bool

$\leq_{\mathsf{int2}}$ computes the less-than-or-equal-relation on integers and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2}\\ \quad (\mathsf{x} \leq_{\mathsf{int2}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow ((\mathsf{x} <_{\mathsf{int2}} \mathsf{y}) \equiv \mathsf{true} \vee \mathsf{x} \equiv \mathsf{y})\end{array}$$

Since this a non-recursive constructive specification, we are done.

4.8 $>_{\mathsf{int2}}$: int2 × int2 → bool

$>_{\mathsf{int2}}$ computes the greater-than-relation on integers and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2}\\ \quad (\mathsf{x} >_{\mathsf{int2}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow (\mathsf{y} <_{\mathsf{int2}} \mathsf{x}) \equiv \mathsf{true}\end{array}$$

Since this a non-recursive constructive specification, we are done.

4.9 $\geq_{\mathsf{int2}}$: int2 × int2 → bool

$\geq_{\mathsf{int2}}$ computes the greater-than-or-equal-relation on integers and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{int2}\\ \quad (\mathsf{x} \geq_{\mathsf{int2}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow (\mathsf{y} \leq_{\mathsf{int2}} \mathsf{x}) \equiv \mathsf{true}\end{array}$$

Since this a non-recursive constructive specification, we are done.

5 Finite Lists, list

This specification of finite lists (of nats), list, uses two constructor functions $\mathsf{nil} :\rightarrow \mathsf{list}$, generating the empty list, and $\mathsf{cons} : \mathsf{nat} \times \mathsf{list} \rightarrow \mathsf{list}$, inserting an element into a list. Equality on lists is specified by the axioms:

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list}$$
$$\mathsf{nil} \not\equiv \mathsf{cons}(\mathsf{x}, \mathsf{A}) \text{ and}$$

$$\forall\, \mathsf{x}, \mathsf{y}{:}\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{list}$$
$$\mathsf{cons}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{cons}(\mathsf{y}, \mathsf{B}) \rightarrow (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{A} \equiv \mathsf{B}).$$

By the above specification we have defined a freely generated data type. Hence, the constructor function cons is size increasing, and we can synthesize the strictness predicate $\Theta^2_{\mathsf{cons}} : \mathsf{nat} \times \mathsf{list} \rightarrow \mathsf{bool}$ and the minimal representation predicate $\Gamma_{\mathsf{cons}} : \mathsf{nat} \times \mathsf{list} \rightarrow \mathsf{bool}$ by

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list}$$
$$\Theta^2_{\mathsf{cons}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true} \text{ and}$$

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list}$$
$$\Gamma_{\mathsf{cons}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true}.$$

Furthermore, the constructor functions of list are non-overlapping, which leads to the following synthesis of the destructor functions $\mathsf{car} : \mathsf{list} \rightarrow \mathsf{nat}$ for the first argument of the constructor function cons and $\mathsf{cdr} : \mathsf{list} \rightarrow \mathsf{list}$ for the second argument of the constructor function cons:

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{list}$$
$$\mathsf{A} \equiv \mathsf{cons}(\mathsf{x}, \mathsf{B}) \rightarrow \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})),$$

$$\mathsf{car}(\mathsf{nil}) \equiv 0\ (\equiv \triangledown_{\mathsf{nat}}) \land \mathsf{cdr}(\mathsf{nil}) \equiv \mathsf{nil}, \text{ and}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{x}, \mathsf{B}) \to \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

Furthermore, cdr is 1-bounded with difference predicate $\Delta^1_{\mathsf{cdr}} : \mathsf{list} \to \mathsf{bool}$:

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})).\end{array}$$

For the data type list we will give constructive function and predicate specifications for app, member, min, max, length, delete, last, butlast, sort, $<_{\mathsf{list}}$, $\leq_{\mathsf{list}}$, $>_{\mathsf{list}}$, and $\geq_{\mathsf{list}}$.

5.1 app : list × list → list

app computes the concatenation of two lists and is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{app}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{B}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \to \mathsf{app}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{app}(\mathsf{cdr}(\mathsf{A}), \mathsf{B}))\end{array}$$

The recursion ordering of app is well-founded. There is only one definition case with a single recursive call of app. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))$$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \to (\mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})).\end{array}$$

5.2 $\mathsf{member} : \mathsf{nat} \times \mathsf{list} \to \mathsf{bool}$

member computes the containment relation of a natural number in a list and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{member}(\mathsf{x},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{car}(\mathsf{A}))\\ \qquad \to \mathsf{member}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))\\ \qquad \to \mathsf{member}(\mathsf{x},\mathsf{A}) \equiv \mathsf{member}(\mathsf{x},\mathsf{cdr}(\mathsf{A}))\end{array}$$

The recursion ordering of member is well-founded. There is only one definition case with a single recursive call of member. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))$$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A})) \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A})) \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})).\end{array}$$

5.3 $\mathsf{length} : \mathsf{list} \to \mathsf{nat}$

length computes the length of a list and is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}{:}\,\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{length}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}{:}\,\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \to \mathsf{length}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{length}(\mathsf{cdr}(\mathsf{A})))\end{array}$$

The recursion ordering of length is well-founded. There is only one definition case with a single recursive call of length. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$\mathsf{A \equiv cons(car(A), cdr(A))}$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr(A)} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$\forall\ \mathsf{A}{:}\,\mathsf{list}$
$\quad \mathsf{A \equiv cons(car(A), cdr(A))} \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})$

which can be simplified to

$\forall\ \mathsf{A}{:}\,\mathsf{list}$
$\quad \mathsf{A \equiv cons(car(A), cdr(A))} \rightarrow \mathsf{A \equiv cons(car(A), cdr(A))}.$

5.4 delete : nat × list → list

delete computes the delete operation on lists, thus it removes the first occurrence of a specified natural number in a list, and it is defined by:

$\forall\ \mathsf{x}{:}\,\mathsf{nat}\ \forall\ \mathsf{A}{:}\,\mathsf{list}$
$\quad \mathsf{A \equiv nil} \rightarrow \mathsf{delete(x, A) \equiv nil}$

$\forall\ \mathsf{x}{:}\,\mathsf{nat}\ \forall\ \mathsf{A}{:}\,\mathsf{list}$
$\quad (\mathsf{A \equiv cons(car(A), cdr(A))} \wedge \mathsf{x \equiv car(A)})$
$\qquad \rightarrow \mathsf{delete(x, A) \equiv cdr(A)}$

$\forall\ \mathsf{x}{:}\,\mathsf{nat}\ \forall\ \mathsf{A}{:}\,\mathsf{list}$
$\quad (\mathsf{A \equiv cons(car(A), cdr(A))} \wedge \mathsf{x \not\equiv car(A)})$
$\qquad \rightarrow \mathsf{delete(x, A) \equiv cons(car(A), delete(x, cdr(A)))}$

The recursion ordering of delete is well-founded. There is only one definition case with a single recursive call of delete. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$(\mathsf{A \equiv cons(car(A), cdr(A))} \wedge \mathsf{x \not\equiv car(A)})$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr(A)} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A})) \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A})) \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})).\end{array}$$

In addition, delete denotes a 2-bounded function symbol. To prove this property, first of all, we need to show that delete is completely specified, by:

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{nil} \vee\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{car}(\mathsf{A})) \vee\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))\end{array}$$

Then, we examine each definition case separately. For the first case we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \mathsf{A} \equiv \mathsf{nil}, \mathsf{nil} \preceq_{\mathsf{list}} \mathsf{nil}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \mathsf{A} \equiv \mathsf{nil}, \mathsf{nil} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Equation\ 1}$$

For the second case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{car}(\mathsf{A}))$$

by φ, and we obtain the derivation in the Estimation Calculus:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

And, for the third case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))$$

by φ. Furthermore, since this case is recursive, we can assume an additional inference rule as the induction hypothesis:

$$\xi \Rightarrow \cfrac{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \Delta\rangle}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \mathsf{Induction\ Hypothesis}$$

where ξ is an abbreviation for the formula

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list}\ \varphi \to \Delta$$

Then, we obtain the derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{list}} \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})), \\ \Delta^2_{\mathsf{delete}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{list}} \mathsf{A}, \\ \Delta^2_{\mathsf{delete}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where in order to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and to allow the application of the Weak Embedding Rule,

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list}\ \varphi \to \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}$$

has to be shown.

In order to synthesize the difference predicate $\Delta^2_{\mathsf{delete}} : \mathsf{nat} \times \mathsf{list} \to \mathsf{bool}$, we use the simplified difference formulas from each derivation, and we obtain:

$$\begin{array}{l} \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list} \\ \quad \mathsf{A} \equiv \mathsf{nil} \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{car}(\mathsf{A})) \\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A})) \\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{cdr}(\mathsf{A})) \end{array}$$

5.5 min : list → nat

min computes the minimal element in a non-empty list, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{list} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil}) \\ \qquad \to \mathsf{min}(\mathsf{A}) \equiv \mathsf{car}(\mathsf{A}) \end{array}$$

$$\forall\, \mathsf{A}\!:\!\mathsf{list}\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix} \to \mathsf{min}(\mathsf{A}) \equiv \mathsf{min}(\mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))$$

$$\forall\, \mathsf{A}\!:\!\mathsf{list}\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix} \to \mathsf{min}(\mathsf{A}) \equiv \mathsf{min}(\mathsf{cdr}(\mathsf{A}))$$

The recursion ordering of min is well-founded. There are two definition cases with one recursive call in each. For the first recursive case we obtain the derivation in the Estimation Calculus, abbreviating the case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix}$$

by φ:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{A}), \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \begin{matrix} \varphi, \mathsf{cdr}(\mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{A}), \\ \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \end{matrix} \right\rangle}\ \mathsf{Estimation}}{\left\langle \begin{matrix} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{list}} \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})), \\ \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{matrix} \right\rangle}\ \mathsf{Weak\ Embedding}}{\left\langle \begin{matrix} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{list}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{matrix} \right\rangle}\ \mathsf{Equation\ 3}$$

where in order to enable the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}\!:\!\mathsf{list}\ \varphi \to \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}$$

has to be shown.

To ensure the strict relation, we, therefore, need to prove

$$\forall\, \mathsf{A}\!:\!\mathsf{list}\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix} \to \begin{pmatrix} \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge\\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right)\\ \quad\to \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))).\end{array}$$

For the second recursive case we abbreviate the case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge\\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)$$

by φ, and we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge\\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\\ \quad\to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge\\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\\ \quad\to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})).\end{array}$$

5.6 max : list → nat

max computes the maximal element in a non-empty list, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil})\\ \quad\to \mathsf{max}(\mathsf{A}) \equiv \mathsf{car}(\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge\\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right)\\ \quad\to \mathsf{max}(\mathsf{A}) \equiv \mathsf{max}(\mathsf{cdr}(\mathsf{A}))\end{array}$$

$$\forall\, \mathsf{A}\!:\!\mathsf{list} \\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix} \\ \rightarrow \mathsf{max}(\mathsf{A}) \equiv \mathsf{max}(\mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))$$

The recursion ordering of max is well-founded. There are two definition cases with one recursive call in each. For the second recursive case we obtain the derivation in the Estimation Calculus, abbreviating the case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix}$$

by φ:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{A}), \mathsf{false} \rangle}\,\text{Identity}}{\left\langle \begin{matrix} \varphi, \mathsf{cdr}(\mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{A}), \\ \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \end{matrix} \right\rangle}\,\text{Estimation}}{\left\langle \begin{matrix} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{list}} \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})), \\ \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{matrix} \right\rangle}\,\text{Weak Embedding}}{\left\langle \begin{matrix} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{list}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{matrix} \right\rangle}\,\text{Equation 3}$$

where in order to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}\!:\!\mathsf{list}\ \varphi \rightarrow \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}$$

has to be shown.

In order to ensure the strict relation, we, therefore, need to prove

$$\forall\, \mathsf{A}\!:\!\mathsf{list} \\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix} \\ \rightarrow \begin{pmatrix} \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix}$$

which can be simplified to

$$\forall\, \mathsf{A}\!:\!\mathsf{list} \\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix} \\ \rightarrow \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))).$$

For the first recursive case we abbreviate the case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix}$$

by φ, and we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\,\mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\,\mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{list} \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix} \\ \quad\quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{list} \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix} \\ \quad\quad \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})). \end{array}$$

5.7 last : list → nat

last computes the last element in a non-empty list, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{list} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil}) \\ \quad\quad \to \mathsf{last}(\mathsf{A}) \equiv \mathsf{car}(\mathsf{A}) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{list} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))) \\ \quad\quad \to \mathsf{last}(\mathsf{A}) \equiv \mathsf{last}(\mathsf{cdr}(\mathsf{A})) \end{array}$$

The recursion ordering of last is well-founded. There is only one definition case with a single recursive call. Hence, we use the Estimation Calculus, abbreviating the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))$$

by φ. We obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\;\mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\;\mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))\\ \qquad \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})).\end{array}$$

5.8 butlast : list → list

butlast computes the original list without its last element, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{butlast}(\mathsf{A}) \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil})\\ \qquad \to \mathsf{butlast}(\mathsf{A}) \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))\\ \qquad \to \mathsf{butlast}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{butlast}(\mathsf{cdr}(\mathsf{A})))\end{array}$$

The recursion ordering of butlast is well-founded. There is only one definition case with a single recursive call. Hence, we use the Estimation Calculus, abbreviating the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))$$

by φ. We obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\;\mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\;\mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))\\ \qquad \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})).\end{array}$$

To prove that butlast is 1-bounded, first of all, we need to show that butlast is completely specified, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list}\\ \quad \mathsf{A} \equiv \mathsf{nil} \vee\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil}) \vee\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))\end{array}$$

Next, we examine each definition case separately. For the first definition case we abbreviate the case condition

$$\mathsf{A} \equiv \mathsf{nil}$$

by φ. Then, we obtain the derivation:

$$\begin{array}{c} - \\ \text{——— Identity ———} \\ \langle \varphi, \mathsf{nil} \preceq_{\mathsf{list}} \mathsf{nil}, \mathsf{false}\rangle \\ \text{—— Equation 1 ——} \\ \langle \varphi, \mathsf{nil} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle \end{array}$$

For the second definition case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil})$$

by φ, and we obtain the derivation:

$$\begin{array}{c} - \\ \text{———— Strong Estimation ————} \\ \langle \varphi, \mathsf{nil} \preceq_{\mathsf{list}} \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})), \mathsf{true}\rangle \\ \text{————— Equation 1 —————} \\ \langle \varphi, \mathsf{nil} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{true}\rangle \end{array}$$

For the third definition case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))$$

by φ. Since this is a recursive definition case, we may assume the additional inference rule

$$\xi \Rightarrow \quad \begin{array}{c} \langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \Delta\rangle \\ \text{——————— Induction Hypothesis ———————} \\ \langle \varphi, \mathsf{butlast}(\mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{A}), \Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\rangle \end{array}$$

where ξ is an abbreviation for the formula

$$\forall\, \mathsf{A}\!:\!\mathsf{list}\ \varphi \to \Delta$$

as an induction hypothesis. Now, we obtain the derivation:

$$\begin{array}{c}
-\\
\text{——————— Identity ———————}\\
\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle\\
\text{——————— Estimation ———————}\\
\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle\\
\left\langle \begin{array}{c}
\text{——————— Induction Hypothesis ———————}\\
\langle \varphi, \mathsf{butlast}(\mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{A}), \Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\rangle\\
\text{——————— Weak Embedding ———————}
\end{array}\right\rangle\\
\left\langle \begin{array}{c}
\varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{list}} \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})),\\
\Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false}
\end{array}\right\rangle\\
\text{——————— Equation 3 ———————}\\
\left\langle \begin{array}{c}
\varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{list}} \mathsf{A},\\
\Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false}
\end{array}\right\rangle
\end{array}$$

where in order to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}\!:\!\mathsf{list}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and to allow the application of the Weak Embedding Rule,

$$\forall\, \mathsf{A}\!:\!\mathsf{list}\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}$$

has to be shown.

Using the simplified difference formulas, we can now synthesize the definition of $\Delta^1_{\mathsf{butlast}} : \mathsf{list} \to \mathsf{bool}$:

$$\begin{array}{l}
\forall\, \mathsf{A}\!:\!\mathsf{list}\\
\quad \mathsf{A} \equiv \mathsf{nil} \to \Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{false}
\end{array}$$

$$\begin{array}{l}
\forall\, \mathsf{A}\!:\!\mathsf{list}\\
\quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil})\\
\qquad \to \Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{true}
\end{array}$$

$$\begin{array}{l}
\forall\, \mathsf{A}\!:\!\mathsf{list}\\
\quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))\\
\qquad \to \Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A}))
\end{array}$$

5.9 sort : list → list

sort sorts a list, defined by:

$$\begin{array}{l}
\forall\, \mathsf{A}\!:\!\mathsf{list}\\
\quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{sort}(\mathsf{A}) \equiv \mathsf{nil}
\end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{list} \\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \\ \qquad \to \mathsf{sort}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{min}(\mathsf{A}),\mathsf{sort}(\mathsf{delete}(\mathsf{min}(\mathsf{A}),\mathsf{A}))) \end{array}$$

The recursion ordering of sort is well-founded. There is only one recursive definition case with a single recursive call. Hence, we abbreviate the case condition

$$\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))$$

by φ. Using the Estimation Calculus, we obtain the following derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\,\text{Identity}}{\langle \varphi, \mathsf{delete}(\mathsf{min}(\mathsf{A}),\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{min}(\mathsf{A}),\mathsf{A}) \equiv \mathsf{true}\rangle}\,\text{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{list} \\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \to (\mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{min}(\mathsf{A}),\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be proved by induction.

5.10 $<_{\mathsf{list}}$: list $\times$ list $\to$ bool

$<_{\mathsf{list}}$ computes the less-than-relation on lists, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A},\mathsf{B}:\mathsf{list} \\ \quad \mathsf{B} \equiv \mathsf{nil} \to (\mathsf{A} <_{\mathsf{list}} \mathsf{B}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A},\mathsf{B}:\mathsf{list} \\ \quad (\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{nil}) \\ \qquad \to (\mathsf{A} <_{\mathsf{list}} \mathsf{B}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A},\mathsf{B}:\mathsf{list} \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{array}\right) \\ \qquad \to (\mathsf{A} <_{\mathsf{list}} \mathsf{B}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A},\mathsf{B}:\mathsf{list} \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{array}\right) \\ \qquad \to (\mathsf{A} <_{\mathsf{list}} \mathsf{B}) \equiv \mathsf{false} \end{array}$$

The recursion ordering of $<_{\mathsf{list}}$ is well-founded: There are two definition cases with a single recursive call of $<_{\mathsf{list}}$ in each. For each recursive definition case and each argument we use the Estimation Calculus. Starting with the first recursive case, we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right)$$

by φ. For the first argument of $<_{\mathsf{list}}$, A, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{list}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right) \\ \quad\ \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of Δ^1_{cdr} to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right) \\ \quad\ \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})).\end{array}$$

And for the second argument of $<_{\mathsf{list}}$, B, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{B} \preceq_{\mathsf{list}} \mathsf{B}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{B}) \preceq_{\mathsf{list}} \mathsf{B}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{B}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{list}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right) \\ \quad\ \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{B}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of Δ^1_{cdr} to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right) \\ \quad\ \to \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})).\end{array}$$

For the second recursive definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right)$$

by φ. For the first argument of $<_{\mathsf{list}}$, A, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{list}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right) \\ \quad\ \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of Δ^1_{cdr} to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right) \\ \quad\ \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})).\end{array}$$

And for the second argument of $<_{\mathsf{list}}$, B, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{B} \preceq_{\mathsf{list}} \mathsf{B}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{B}) \preceq_{\mathsf{list}} \mathsf{B}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{B}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{list}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right) \\ \quad\ \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{B}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of Δ^1_{cdr} to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right) \\ \quad\ \to \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})).\end{array}$$

Thus, the recursion ordering of $<_{\mathsf{list}}$ is a well-founded ordering.

In addition, $<_{\mathsf{list}}$ denotes a well-founded ordering as well. To prove that, we first have to show that $<_{\mathsf{list}}$ is completely specified, i.e.,

$$\begin{array}{l}
\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\
\quad (\mathsf{B} \equiv \mathsf{nil}) \vee \\
\quad (\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{nil}) \vee \\
\quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{array} \right) \vee \\
\quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{array} \right)
\end{array}$$

Next, for each definition case we show that

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list}\ (\mathsf{A} <_{\mathsf{list}} \mathsf{B}) \equiv \mathsf{true} \to \mathsf{A} \prec_{\mathsf{list}} \mathsf{B},$$

again, using the Estimation Calculus. For the first case we obtain

$$\frac{-}{\langle \mathsf{B} \equiv \mathsf{nil} \wedge \mathsf{false} \equiv \mathsf{true}, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{B}, \Delta_1 \rangle}\ \mathsf{Tautology}$$

where in order to enable the application of the Tautology Rule, the first-order formula

$$\begin{array}{l}
\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\
\quad \neg(\mathsf{B} \equiv \mathsf{nil} \wedge \mathsf{false} \equiv \mathsf{true})
\end{array}$$

has to be proved. To prove the strict relation, the formula

$$\begin{array}{l}
\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\
\quad (\mathsf{B} \equiv \mathsf{nil} \wedge \mathsf{false} \equiv \mathsf{true}) \to \Delta_1
\end{array}$$

has to be shown. For the second case we obtain the derivation

$$\begin{array}{c}
- \\
\hline \text{Strong Estimation} \\
\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{true} \equiv \mathsf{true}, \\ \mathsf{nil} \preceq_{\mathsf{list}} \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})), \mathsf{true} \end{array} \right\rangle \\
\hline \text{Equation 1} \\
\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{true} \equiv \mathsf{true}, \\ \mathsf{nil} \preceq_{\mathsf{list}} \mathsf{B}, \mathsf{true} \end{array} \right\rangle \\
\hline \text{Equation 5} \\
\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{true} \equiv \mathsf{true}, \\ \mathsf{A} \preceq_{\mathsf{list}} \mathsf{B}, \mathsf{true} \end{array} \right\rangle
\end{array}$$

showing the strict relation by

$$\begin{array}{l}
\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\
\quad (\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{true} \equiv \mathsf{true}) \\
\qquad \to \mathsf{true}
\end{array}$$

The third definition case is a recursive case. Hence, we need to make an additional case analysis:

$(\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}$ or

$(\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}$.

For the first case we can assume as an induction hypothesis the inference rule:

$$\frac{-}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{B}), \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where we use φ as an abbreviation for

$$\left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{array}\right)$$

Then, the derivation of

$\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{B}, \Delta_3\rangle$

is achieved by:

$$\cfrac{\cfrac{\cfrac{\cfrac{-}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{cdr}(\mathsf{B}), \mathsf{true}\rangle}\ \text{Induction Hypothesis}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{list}} \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})), \\ \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false} \end{array}\right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{list}} \mathsf{B}, \\ \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false} \end{array}\right\rangle}\ \text{Equation 3}}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{B}, \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false}\rangle}\ \text{Equation 5}$$

where in order to enable the application of the Weak Embedding Rule, the first-order formula

$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list}\ \varphi \to \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}$

has to be shown. The strict relation is proved by

$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list}$
$\quad \varphi \to (\mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false})$.

For the second case we cannot assume an induction hypothesis. We prove the estimation formula

$$\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}, \\ \mathsf{A} \preceq_{\mathsf{list}} \mathsf{B}, \Delta_4 \end{array}\right\rangle$$

by an application of the Tautology Rule, where in order to enable this application, it is necessary to prove the first-order formula:

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \neg \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{array} \right)$$

To prove the strict relation, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{array} \right) \\ \to \Delta_4$$

needs to be shown.

The fourth definition case is also a recursive case. Hence, we need to make an additional case analysis:

$(\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}$ or

$(\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}$.

Although for the first case we could assume an induction hypothesis, this is not necessary since the derivation of the estimation formula

$$\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}, \\ \mathsf{A} \preceq_{\mathsf{list}} \mathsf{B}, \Delta_5 \end{array} \right\rangle$$

can be achieved by the application of the Tautology Rule. In order to enable this application, the first-order formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \neg \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{array} \right)$$

has to be shown. And for the strict relation, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{array} \right) \\ \to \Delta_5$$

needs to be proved. For the second case we cannot assume an induction hypothesis. We prove the estimation formula

$$\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true} \wedge (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{list}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}, \\ \mathsf{A} \preceq_{\mathsf{list}} \mathsf{B}, \Delta_6 \end{array} \right\rangle$$

by an application of the Tautology Rule, where in order to enable this application, it is necessary to prove the first-order formula:

$$\forall\,\mathsf{A},\mathsf{B}:\mathsf{list}\;\; \neg\left(\begin{array}{c}\mathsf{B}\equiv\mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ (\mathsf{cdr}(\mathsf{A})<_{\mathsf{list}}\mathsf{cdr}(\mathsf{B}))\equiv\mathsf{false}\wedge\mathsf{false}\equiv\mathsf{true}\wedge(\mathsf{cdr}(\mathsf{A})<_{\mathsf{list}}\mathsf{cdr}(\mathsf{B}))\equiv\mathsf{false}\end{array}\right)$$

To prove the strict relation, the formula

$$\forall\,\mathsf{A},\mathsf{B}:\mathsf{list}\;\; \left(\begin{array}{c}\mathsf{B}\equiv\mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ (\mathsf{cdr}(\mathsf{A})<_{\mathsf{list}}\mathsf{cdr}(\mathsf{B}))\equiv\mathsf{false}\wedge\mathsf{false}\equiv\mathsf{true}\wedge(\mathsf{cdr}(\mathsf{A})<_{\mathsf{list}}\mathsf{cdr}(\mathsf{B}))\equiv\mathsf{false}\end{array}\right)\rightarrow\Delta_6$$

needs to be shown.

Having proved all these obligations, $<_{\mathsf{list}}$ denotes a well-founded order relation.

5.11 $\leq_{\mathsf{list}}$: list × list → bool

$\leq_{\mathsf{list}}$ computes the less-than-or-equal-relation on lists, and it is defined by:

$$\forall\,\mathsf{A},\mathsf{B}:\mathsf{list}\;\; (\mathsf{A}\leq_{\mathsf{list}}\mathsf{B})\equiv\mathsf{true}\leftrightarrow(\mathsf{B}<_{\mathsf{list}}\mathsf{A})\equiv\mathsf{false}$$

Since this is a non-recursive constructive definition we are done.

5.12 $>_{\mathsf{list}}$: list × list → bool

$>_{\mathsf{list}}$ computes the greater-than-relation on lists, and it is defined by:

$$\forall\,\mathsf{A},\mathsf{B}:\mathsf{list}\;\; (\mathsf{A}>_{\mathsf{list}}\mathsf{B})\equiv\mathsf{true}\leftrightarrow(\mathsf{B}<_{\mathsf{list}}\mathsf{A})\equiv\mathsf{true}$$

Since this is a non-recursive constructive definition we are done.

5.13 $\geq_{\mathsf{list}}$: list × list → bool

$\geq_{\mathsf{list}}$ computes the greater-than-relation on lists, and it is defined by:

$$\forall\,\mathsf{A},\mathsf{B}:\mathsf{list}\;\; (\mathsf{A}\geq_{\mathsf{list}}\mathsf{B})\equiv\mathsf{true}\leftrightarrow(\mathsf{B}\leq_{\mathsf{list}}\mathsf{A})\equiv\mathsf{true}$$

Since this is a non-recursive constructive definition we are done.

6

Finite Lists, list2

This specification of finite lists (of nats), list2, uses three constructor functions nil $:\to$ list2, generating the empty list, single : nat $\to$ list2, generating a singleton list, and app : list2 $\times$ list2 $\to$ list2, concatenating two lists. Equality on list2 is specified by the axioms:

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\\ \quad \mathsf{nil} \not\equiv \mathsf{single}(\mathsf{x})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}:\mathsf{list2}\\ \quad \mathsf{nil} \equiv \mathsf{app}(\mathsf{A},\mathsf{B}) \leftrightarrow (\mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{B} \equiv \mathsf{nil})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}:\mathsf{list2}\\ \quad \mathsf{single}(\mathsf{x}) \equiv \mathsf{app}(\mathsf{A},\mathsf{B})\\ \qquad \leftrightarrow ((\mathsf{A} \equiv \mathsf{single}(\mathsf{x}) \wedge \mathsf{B} \equiv \mathsf{nil}) \vee (\mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{B} \equiv \mathsf{single}(\mathsf{x})))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}:\mathsf{nat}\\ \quad \mathsf{single}(\mathsf{x}) \equiv \mathsf{single}(\mathsf{y}) \leftrightarrow \mathsf{x} \equiv \mathsf{y}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list2}\\ \quad \mathsf{app}(\mathsf{nil},\mathsf{A}) \equiv \mathsf{A}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}:\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}:\mathsf{list2}\\ \quad \mathsf{app}(\mathsf{single}(\mathsf{x}),\mathsf{A}) \equiv \mathsf{app}(\mathsf{single}(\mathsf{y}),\mathsf{B}) \leftrightarrow (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{A} \equiv \mathsf{B})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B},\mathsf{C}:\mathsf{list2}\\ \quad \mathsf{app}(\mathsf{app}(\mathsf{A},\mathsf{B}),\mathsf{C}) \equiv \mathsf{app}(\mathsf{A},\mathsf{app}(\mathsf{B},\mathsf{C}))\end{array}$$

By the above specification we have defined a non-freely generated data type. Hence, we must prove the constructor function app to be size increasing by using the respective implementation specification. Furthermore, the strictness predicates $\Theta^1_{\mathsf{app}} : \mathsf{list2} \times \mathsf{list2} \to \mathsf{bool}$ and $\Theta^2_{\mathsf{app}} : \mathsf{list2} \times \mathsf{list2} \to \mathsf{bool}$, as well as the minimal representation predicate $\Gamma_{\mathsf{app}} : \mathsf{list2} \times \mathsf{list2} \to \mathsf{bool}$ have to be synthesized.

The implementation specification is automatically generated using the constructor functions $\mathsf{nil}_{\mathrm{I}} :\to \mathsf{list2}_{\mathrm{I}}$, $\mathsf{single}_{\mathrm{I}} : \mathsf{nat} \to \mathsf{list2}_{\mathrm{I}}$, $\mathsf{app}_{\mathrm{I}} : \mathsf{list2}_{\mathrm{I}} \times \mathsf{list2}_{\mathrm{I}} \to \mathsf{list2}_{\mathrm{I}}$, and the new equality predicate $\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}} : \mathsf{list2}_{\mathrm{I}} \times \mathsf{list2}_{\mathrm{I}} \to \mathsf{bool}$.

$\forall\, \mathsf{x} : \mathsf{nat}$
$\quad \mathsf{nil}_{\mathrm{I}} \not\equiv \mathsf{single}_{\mathrm{I}}(\mathsf{x})$

$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{nil}_{\mathrm{I}} \not\equiv \mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})$

$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{single}_{\mathrm{I}}(\mathsf{x}) \not\equiv \mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}$
$\quad \mathsf{single}_{\mathrm{I}}(\mathsf{x}) \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{y}) \to \mathsf{x} \equiv \mathsf{y}$

$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{C}, \mathsf{D}) \to (\mathsf{A} \equiv \mathsf{C} \wedge \mathsf{B} \equiv \mathsf{D})$

$\forall\, \mathsf{x} : \mathsf{nat}$
$\quad \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{nil}_{\mathrm{I}}, \mathsf{single}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{false}$

$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{nil}_{\mathrm{I}}, \mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})) \equiv \mathsf{true}$
$\qquad \leftrightarrow (\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{true})$

$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{single}_{\mathrm{I}}(\mathsf{x}), \mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})) \equiv \mathsf{true}$
$$\leftrightarrow \left(\begin{array}{c} (\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{single}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{true}) \vee \\ (\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{single}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{true}) \end{array} \right)$$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}$
$\quad \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{single}_{\mathrm{I}}(\mathsf{x}), \mathsf{single}_{\mathrm{I}}(\mathsf{y})) \equiv \mathsf{true} \leftrightarrow \mathsf{x} \equiv \mathsf{y}$

$\forall\, \mathsf{A} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{app}_{\mathrm{I}}(\mathsf{nil}_{\mathrm{I}}, \mathsf{A}), \mathsf{A}) \equiv \mathsf{true}$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{app}_{\mathrm{I}}(\mathsf{single}_{\mathrm{I}}(\mathsf{x}), \mathsf{A}), \mathsf{app}_{\mathrm{I}}(\mathsf{single}_{\mathrm{I}}(\mathsf{y}), \mathsf{B})) \equiv \mathsf{true}$
$\qquad \leftrightarrow (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true})$

$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{app}_{\mathrm{I}}(\mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}), \mathsf{C}), \mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{app}_{\mathrm{I}}(\mathsf{B}, \mathsf{C}))) \equiv \mathsf{true}$

$\forall\, \mathsf{A} : \mathsf{list2}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{A}) \equiv \mathsf{true}$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2_I}$$
$$\mathsf{Eq_{list2_I}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \to \mathsf{Eq_{list2_I}}(\mathsf{B}, \mathsf{A}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2_I}$$
$$(\mathsf{Eq_{list2_I}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \wedge \mathsf{Eq_{list2_I}}(\mathsf{B}, \mathsf{C}) \equiv \mathsf{true})$$
$$\to \mathsf{Eq_{list2_I}}(\mathsf{A}, \mathsf{C}) \equiv \mathsf{true}$$

Since $\mathsf{list2_I}$ is freely generated, the strictness predicates[1] $\theta^1_{\mathsf{app_I}} : \mathsf{list2_I} \times \mathsf{list2_I} \to \mathsf{bool}$ and $\theta^2_{\mathsf{app_I}} : \mathsf{list2_I} \times \mathsf{list2_I} \to \mathsf{bool}$, as well as the minimal representation predicate $\gamma_{\mathsf{app_I}} : \mathsf{list2_I} \times \mathsf{list2_I} \to \mathsf{bool}$ are defined by:

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2_I}$$
$$\theta^1_{\mathsf{app_I}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2_I}$$
$$\theta^2_{\mathsf{app_I}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2_I}$$
$$\gamma_{\mathsf{app_I}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

In addition, all constructor functions of $\mathsf{list2_I}$ are non-overlapping. Hence, the destructor function $\mathsf{get\_nat_I} : \mathsf{list2_I} \to \mathsf{nat}$ for the constructor function $\mathsf{single_I}$ is defined by

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2_I}$$
$$\mathsf{A} \equiv \mathsf{single_I}(\mathsf{x}) \to \mathsf{A} \equiv \mathsf{single_I}(\mathsf{get\_nat_I}(\mathsf{A})),$$

$$\mathsf{get\_nat_I}(\mathsf{nil_I}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2_I}$$
$$\mathsf{get\_nat_I}(\mathsf{app_I}(\mathsf{A}, \mathsf{B})) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}})$$

And for the constructor function $\mathsf{app_I}$ we introduce two destructor functions $\mathsf{left\_list_I} : \mathsf{list2_I} \to \mathsf{list2_I}$ for the first argument of $\mathsf{app_I}$ and $\mathsf{right\_list_I} : \mathsf{list2_I} \to \mathsf{list2_I}$ for the second argument of $\mathsf{app_I}$. For these destructor functions we obtain the following representation axioms:

$$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2_I}$$
$$\mathsf{A} \equiv \mathsf{app_I}(\mathsf{B}, \mathsf{C}) \to \mathsf{A} \equiv \mathsf{app_I}(\mathsf{left\_list_I}(\mathsf{A}), \mathsf{right\_list_I}(\mathsf{A}))$$

$$\mathsf{left\_list_I}(\mathsf{nil_I}) \equiv \mathsf{nil_I}$$

$$\mathsf{right\_list_I}(\mathsf{nil_I}) \equiv \mathsf{nil_I}$$

$$\forall\, \mathsf{x} : \mathsf{nat}$$
$$\mathsf{left\_list_I}(\mathsf{single_I}(\mathsf{x})) \equiv \mathsf{single_I}(\mathsf{x})$$

$$\forall\, \mathsf{x} : \mathsf{nat}$$
$$\mathsf{right\_list_I}(\mathsf{single_I}(\mathsf{x})) \equiv \mathsf{single_I}(\mathsf{x})$$

[1]We will denote the strictness predicates for the reflexive constructor functions of the implementation data type by θ, as opposed to the strictness predicates, Θ, for the reflexive constructor functions of the original data type. Similarly, γ shall denote the minimal representation predicate for the implementation constructor function, and Γ for the original constructor function.

$$\forall\ \mathsf{A},\mathsf{B},\mathsf{C}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{B},\mathsf{C}) \to \gamma_{\mathsf{app}_{\mathrm{I}}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A})) \equiv \mathsf{true}$$

Now, $\mathsf{left\_list}_{\mathrm{I}}$ and $\mathsf{right\_list}_{\mathrm{I}}$ are both 1-bounded with difference predicates $\Delta^{\mathrm{I1}}_{\mathsf{left\_list}_{\mathrm{I}}} : \mathsf{list2}_{\mathrm{I}} \to \mathsf{bool}$ and $\Delta^{\mathrm{I2}}_{\mathsf{right\_list}_{\mathrm{I}}} : \mathsf{list2}_{\mathrm{I}} \to \mathsf{bool}$, defined by

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \Delta^{\mathrm{I1}}_{\mathsf{left\_list}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))$$

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \Delta^{\mathrm{I1}}_{\mathsf{right\_list}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))$$

Furthermore, the function $\mathsf{term\_size}_{\mathsf{list2}_{\mathrm{I}}} : \mathsf{list2}_{\mathrm{I}} \to \mathsf{nat}$ is synthesized by:

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{nil}_{\mathrm{I}} \to \mathsf{term\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) \equiv 0$$

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \to \mathsf{term\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) \equiv 0$$

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{term\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) \equiv\\ \qquad\qquad \mathsf{succ}(\mathsf{term\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A})) + \mathsf{term\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{right\_list}_{\mathrm{I}}(\mathsf{A})))$$

In order to have easier proofs, we specify a function $\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}} : \mathsf{list2}_{\mathrm{I}} \to \mathsf{nat}$, by

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{pred}(\mathsf{length}(\mathsf{A})),$$

where $\mathsf{length} : \mathsf{list2}_{\mathrm{I}} \to \mathsf{nat}$ is defined constructively by

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{nil}_{\mathrm{I}} \to \mathsf{length}(\mathsf{A}) \equiv 0$$

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \to \mathsf{length}(\mathsf{A}) \equiv \mathsf{succ}(0)$$

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{length}(\mathsf{A}) \equiv (\mathsf{length}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A})) + \mathsf{length}(\mathsf{right\_list}_{\mathrm{I}}(\mathsf{A})))$$

The specification of length is case-distinct, as proved by

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \neg\Big(\ \mathsf{A} \equiv \mathsf{nil}_{\mathrm{I}} \wedge \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}))\ \Big)$$

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \neg\Big(\ \mathsf{A} \equiv \mathsf{nil}_{\mathrm{I}} \wedge \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))\ \Big)$$

$$\forall\ \mathsf{A}\colon \mathsf{list2}_{\mathrm{I}}\\ \quad \neg\Big(\ \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))\ \Big)$$

Furthermore, the recursion ordering of length is well-founded. To prove that we use the Estimation Calculus. There is only one recursive case with two recursive calls of length. Now, we abbreviate the case condition

$$\mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Then, for the first recursive call the derivation in the Estimation Calculus is given by

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\,\mathsf{Identity}}{\left\langle \varphi, \mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{list2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{left\_list}_{\mathrm{I}}}(\mathsf{A})\right\rangle}\,\mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{aligned}&\forall\, \mathsf{A} : \mathsf{list2}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to (\mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{left\_list}_{\mathrm{I}}}(\mathsf{A}))\end{aligned}$$

which can be simplified to

$$\begin{aligned}&\forall\, \mathsf{A} : \mathsf{list2}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A})).\end{aligned}$$

Similarly, for the second recursive call we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\,\mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{list2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{right\_list}_{\mathrm{I}}}(\mathsf{A})\right\rangle}\,\mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{aligned}&\forall\, \mathsf{A} : \mathsf{list2}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to (\mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{right\_list}_{\mathrm{I}}}(\mathsf{A}))\end{aligned}$$

which can be simplified to

$$\begin{aligned}&\forall\, \mathsf{A} : \mathsf{list2}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to \mathsf{A} \equiv \mathsf{app}_{\mathrm{I}}(\mathsf{left\_list}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_list}_{\mathrm{I}}(\mathsf{A})).\end{aligned}$$

Now, we need to prove that the above axiomatization of $\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}$ computes the minimal size of a list, indeed. Therefore we need to show the following proof obligations

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \to (\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B})) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A} : \mathsf{list2}_{\mathrm{I}}\ \exists\, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \wedge (\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) \geq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B})) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \to \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B})$$

Next, we need to show that app denotes a size increasing constructor function. To do that, we prove:

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$(\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}))) \equiv \mathsf{true} \wedge$$
$$(\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}$$

Finally, we need to define the strictness predicates $\Theta^1_{\mathsf{app}_{\mathrm{I}}} : \mathsf{list2}_{\mathrm{I}} \times \mathsf{list2}_{\mathrm{I}} \to \mathsf{bool}$ and $\Theta^2_{\mathsf{app}_{\mathrm{I}}} : \mathsf{list2}_{\mathrm{I}} \times \mathsf{list2}_{\mathrm{I}} \to \mathsf{bool}$, as well as the minimal representation predicate $\Gamma_{\mathsf{app}_{\mathrm{I}}} : \mathsf{list2}_{\mathrm{I}} \times \mathsf{list2}_{\mathrm{I}} \to \mathsf{bool}$. We suggest the following definitions:

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\Theta^1_{\mathsf{app}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false}$$
$$\leftrightarrow \left(\begin{array}{c} (\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{true}) \vee \\ (\exists\, \mathsf{x} : \mathsf{nat}\ \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{single}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{true}) \end{array} \right)$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\Theta^2_{\mathsf{app}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false}$$
$$\leftrightarrow \left(\begin{array}{c} (\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{true}) \vee \\ (\exists\, \mathsf{x} : \mathsf{nat}\ \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{single}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{true}) \end{array} \right)$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\Gamma_{\mathsf{app}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$
$$\leftrightarrow (\mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{false} \wedge \mathsf{Eq}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{false})$$

However, we have to prove that our suggestions really define the strictness predicates and the minimal representation predicate. Hence, we need to show that

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\Theta^1_{\mathsf{app}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\Theta^2_{\mathsf{app}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}_{\mathrm{I}}$$
$$\Gamma_{\mathsf{app}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$
$$\leftrightarrow \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{app}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{A}) + \mathsf{min\_size}_{\mathsf{list2}_{\mathrm{I}}}(\mathsf{B}))$$

Having done so, we know for our original specification list2 that the constructor function app is size increasing, and we can translate the strictness predicates and the minimal representation predicate into the original specification. Hence, we obtain:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2} \\ \quad \Theta^1_{\mathsf{app}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \leftrightarrow \left(\begin{array}{c} (\mathsf{B} \equiv \mathsf{nil}) \vee \\ (\exists\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{B} \equiv \mathsf{single}(\mathsf{x})) \end{array} \right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2} \\ \quad \Theta^2_{\mathsf{app}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \leftrightarrow \left(\begin{array}{c} (\mathsf{A} \equiv \mathsf{nil}) \vee \\ (\exists\, \mathsf{x} : \mathsf{nat}\ \mathsf{B} \equiv \mathsf{nil} \wedge \mathsf{A} \equiv \mathsf{single}(\mathsf{x})) \end{array} \right)\end{array}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2}\ \Gamma_{\mathsf{app}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{B} \not\equiv \mathsf{nil})$$

The data type $\mathsf{list2}$ possesses overlapping constructor functions, since

$$\mathsf{nil} \equiv \mathsf{app}(\mathsf{nil}, \mathsf{nil})$$

Thus, we cannot use the simplified construction scheme for the destructor functions.

The destructor function $\mathsf{get\_nat} : \mathsf{list2} \to \mathsf{nat}$ for the constructor function single is defined by:

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2} \\ \quad \mathsf{A} \equiv \mathsf{single}(\mathsf{x}) \to \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})),\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{list2} \\ \quad (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})) \to \mathsf{get\_nat}(\mathsf{A}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}}).\end{array}$$

And for the constructor function app we introduce two destructor functions $\mathsf{left\_list} : \mathsf{list2} \to \mathsf{list2}$ for the first argument of app and $\mathsf{right\_list} : \mathsf{list2} \to \mathsf{list2}$ for the second argument of app. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2} \\ \quad \mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \to \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})),\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{list2} \\ \quad (\forall\, \mathsf{B}, \mathsf{C} : \mathsf{list2}\ \mathsf{A} \not\equiv \mathsf{app}(\mathsf{B}, \mathsf{C})) \to (\mathsf{left\_list}(\mathsf{A}) \equiv \mathsf{A} \wedge \mathsf{right\_list}(\mathsf{A}) \equiv \mathsf{A}),\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2} \\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{nil} \wedge (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x}))) \\ \qquad \to \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true},\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2} \\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \wedge (\mathsf{A} \equiv \mathsf{nil} \vee \exists\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \equiv \mathsf{single}(\mathsf{x}))) \\ \qquad \to \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{false},\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2} \\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{nil} \wedge (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})) \wedge \Gamma_{\mathsf{app}}(\mathsf{B}, \mathsf{C}) \equiv \mathsf{true}) \\ \qquad \to \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

They can be simplified to:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2} \\ \quad \mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \to \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})),\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{nil} \wedge (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})))\\ \qquad \rightarrow \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true},\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \wedge (\mathsf{A} \equiv \mathsf{nil} \vee \exists\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \equiv \mathsf{single}(\mathsf{x})))\\ \qquad \rightarrow \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{false}.\end{array}$$

Both reflexive destructor functions of the constructor function app, left_list and right_list, are 1-bounded, and their difference predicates $\Delta^1_{\mathsf{left\_list}} : \mathsf{list2} \rightarrow \mathsf{bool}$ and $\Delta^1_{\mathsf{right\_list}} : \mathsf{list2} \rightarrow \mathsf{bool}$, are defined by

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{list2}\\ \quad \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{list2}\\ \quad \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

For the data type list2 we will give constructive function and predicate specifications for cons, member, length, delete, last, butlast, $<_{\mathsf{list2}}$, $\leq_{\mathsf{list2}}$, $>_{\mathsf{list2}}$, and $\geq_{\mathsf{list2}}$.

6.1 cons : nat × list2 → list2

cons computes the insertion of an element at the beginning of a list, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2}\\ \quad \mathsf{cons}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{app}(\mathsf{single}(\mathsf{x}), \mathsf{A})\end{array}$$

Since this a non-recursive constructive specification, we are done.

6.2 member : nat × list2 → bool

member computes the containment relation of an element in a list and is defined by:

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2}\\ \quad \mathsf{A} \equiv \mathsf{nil} \rightarrow \mathsf{member}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A}))\\ \qquad \rightarrow \mathsf{member}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A}))\\ \qquad \rightarrow \mathsf{member}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \mathsf{member}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad\qquad \leftrightarrow (\mathsf{member}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true} \vee \mathsf{member}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true})\end{array}$$

The recursion ordering of member is well-founded. There is only one definition case with two recursive calls of member. Using the Estimation Calculus, abbreviating the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)$$

by φ, we obtain the following derivation for the first recursive call:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{left\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right).\end{array}$$

A proof of this property is quite simple using the definition of the destructor functions, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}\!:\!\mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B},\mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{nil} \wedge (\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})))\\ \qquad \to \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

Similarly, for the second recursive call of member we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x{:}\,nat}\ \forall\, \mathsf{A{:}\,list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))}\end{array}\right)\\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x{:}\,nat}\ \forall\, \mathsf{A{:}\,list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))}\end{array}\right)\\ \qquad \rightarrow \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \wedge\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list(A), right\_list(A)}) \equiv \mathsf{true}\end{array}\right).\end{array}$$

6.3 length : list2 → nat

length computes the length of a list and is defined by:

$$\begin{array}{l}\forall\, \mathsf{A{:}\,list2}\\ \quad \mathsf{A} \equiv \mathsf{nil} \rightarrow \mathsf{length(A)} \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A{:}\,list2}\\ \quad \mathsf{A} \equiv \mathsf{single(get\_nat(A))} \rightarrow \mathsf{length(A)} \equiv \mathsf{succ(0)}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A{:}\,list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))}\end{array}\right)\\ \qquad \rightarrow \mathsf{length(A)} \equiv (\mathsf{length(left\_list(A))} + \mathsf{length(right\_list(A))})\end{array}$$

The recursion ordering of length is well-founded. There is only one definition case with two recursive calls of length. Using the Estimation Calculus, abbreviating the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))}\end{array}\right)$$

by φ, we obtain the following derivation for the first recursive call:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{left\_list(A)} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x{:}\,nat}\ \forall\, \mathsf{A{:}\,list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))}\end{array}\right)\\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A}))\wedge\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right).\end{array}$$

A proof of this property is quite simple using the definition of the destructor functions, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}{:}\,\mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{nil} \wedge (\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})))\\ \qquad \to \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

Similarly, for the second recursive call of length we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^{1}_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to (\mathsf{false} \vee \Delta^{1}_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A}))\wedge\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right).\end{array}$$

6.4 delete : nat × list2 → list2

delete computes the delete operation on lists, thus it removes the first occurrence of a specified object in a list, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A}))\\ \qquad \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A}))\\ \qquad \rightarrow \mathsf{delete}(\mathsf{x},\mathsf{A}) \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\wedge\\ \mathsf{member}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\\ \qquad \rightarrow \mathsf{delete}(\mathsf{x},\mathsf{A}) \equiv \mathsf{app}(\mathsf{delete}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})),\mathsf{right\_list}(\mathsf{A}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\wedge\\ \mathsf{member}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{false}\end{array}\right)\\ \qquad \rightarrow \mathsf{delete}(\mathsf{x},\mathsf{A}) \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{delete}(\mathsf{x},\mathsf{right\_list}(\mathsf{A})))\end{array}$$

The recursion ordering of delete is well-founded. There are two definition cases with one recursive call of delete in each. Using the Estimation Calculus for the first recursive case, abbreviating the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\wedge\\ \mathsf{member}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)$$

by φ, we obtain the following derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{left\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\wedge\\ \mathsf{member}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\wedge\\ \mathsf{member}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\\ \qquad \rightarrow \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right).\end{array}$$

A proof of this property is quite simple using the definition of the destructor functions, i.e.,

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2} \\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{nil} \wedge (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x}))) \\ \qquad \to \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}. \end{array}$$

For the second recursive definition case of delete we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{member}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{false} \end{pmatrix}$$

by φ, and we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true} \right\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2} \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{member}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{false} \end{pmatrix} \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which simplifies to

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2} \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{member}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{false} \end{pmatrix} \\ \qquad \to \begin{pmatrix} \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \end{pmatrix}. \end{array}$$

Again, we can prove this obligation easily using

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{list2} \\ \quad (\mathsf{A} \equiv \mathsf{app}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{nil} \wedge (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x}))) \\ \qquad \to \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}. \end{array}$$

In addition, delete is a 2-bounded function symbol. To prove this property, first of all, we need to show that delete is completely specified, i.e.,

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{list2} \\ \quad \mathsf{A} \equiv \mathsf{nil} \vee \\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A}) \vee \\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A})) \vee \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{member}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true} \end{pmatrix} \vee \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{member}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{false} \end{pmatrix} \end{array}$$

Then, we examine each definition case separately. For the first case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{nil}$$

by φ, and we obtain

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{nil} \preceq_{\mathsf{list2}} \mathsf{nil}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{nil} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the second definition case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A}))$$

by φ, and we obtain

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{nil} \preceq_{\mathsf{list2}} \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})), \mathsf{false}\rangle}\ \text{Equivalence}}{\langle \varphi, \mathsf{nil} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the third definition case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A}))$$

by φ, and we obtain

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \preceq_{\mathsf{list2}} \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})), \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the fourth definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{member}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true} \end{array}\right)$$

by φ. Since this case is recursive, we may assume an additional inference rule as induction hypothesis:

$$\xi \Rightarrow \dfrac{\langle \varphi, \mathsf{left\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \Delta\rangle}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \preceq_{\mathsf{list2}} \mathsf{left\_list}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where ξ is an abbreviation for the formula

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list2}\ \varphi \to \Delta$$

Using this additional rule, we obtain:

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{left\_list(A)} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{delete(x, left\_list(A))} \preceq_{\mathsf{list2}} \mathsf{left\_list(A)}, \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_list(A)}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where in order to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list2}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_list}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved. On the other hand, we obtain:

$$\dfrac{-}{\langle \varphi, \mathsf{right\_list(A)} \preceq_{\mathsf{list2}} \mathsf{right\_list(A)}, \mathsf{false}\rangle}\ \text{Identity}$$

Hence, we can continue with the following derivation:

$$\dfrac{\dfrac{\begin{array}{c}\langle \varphi, \mathsf{delete(x, left\_list(A))} \preceq_{\mathsf{list2}} \mathsf{left\_list(A)}, \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_list(A)}) \equiv \mathsf{true}\rangle\,, \\ \langle \varphi, \mathsf{right\_list(A)} \preceq_{\mathsf{list2}} \mathsf{right\_list(A)}, \mathsf{false}\rangle\end{array}}{\left\langle \begin{array}{c}\varphi, \mathsf{app(delete(x, left\_list(A)), right\_list(A))} \preceq_{\mathsf{list2}} \mathsf{app(left\_list(A), right\_list(A))}, \\ \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_list(A)}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{app}}(\mathsf{delete(x, left\_list(A))}, \mathsf{right\_list(A)}) \equiv \mathsf{false}\end{array}\right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c}\varphi, \mathsf{app(delete(x, left\_list(A)), right\_list(A))} \preceq_{\mathsf{list2}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_list(A)}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{app}}(\mathsf{delete(x, left\_list(A))}, \mathsf{right\_list(A)}) \equiv \mathsf{false}\end{array}\right\rangle}\ \text{Equation 3}$$

where in order to enable the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{list2}\ \varphi \to \Gamma_{\mathsf{app}}(\mathsf{left\_list(A)}, \mathsf{right\_list(A)}) \equiv \mathsf{true}$$

has to be shown.

Finally, for the fifth definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))} \wedge \\ \mathsf{member(x, left\_list(A))} \equiv \mathsf{false}\end{array}\right)$$

by φ. Since this case is also a recursive, we may assume an additional inference rule as induction hypothesis:

$$\xi \Rightarrow \begin{array}{c} \langle \varphi, \mathsf{right\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \Delta \rangle \\ \text{———————— Induction Hypothesis ————————} \\ \langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \preceq_{\mathsf{list2}} \mathsf{right\_list}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \rangle \end{array}$$

where ξ is an abbreviation for the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list2}\ \varphi \to \Delta$$

Using this additional rule, we obtain:

$$\begin{array}{c} - \\ \text{———— Identity ————} \\ \langle \varphi, \mathsf{left\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{left\_list}(\mathsf{A}), \mathsf{false} \rangle \end{array}$$

On the other hand, we also obtain:

$$\begin{array}{c} - \\ \text{———————— Identity ————————} \\ \langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \rangle \\ \text{———————— Estimation ————————} \\ \left\langle \varphi, \mathsf{right\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true} \right\rangle \\ \text{———————— Induction Hypothesis ————————} \\ \langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \preceq_{\mathsf{list2}} \mathsf{right\_list}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \rangle \end{array}$$

where in order to apply the induction hypothesis, the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list2}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be shown. Hence, we can continue with the following derivation:

$$\begin{array}{c} \langle \varphi, \mathsf{left\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{left\_list}(\mathsf{A}), \mathsf{false} \rangle\ , \\ \langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \preceq_{\mathsf{list2}} \mathsf{right\_list}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \rangle \\ \text{———————— Weak Embedding ————————} \\ \left\langle \begin{array}{c} \varphi, \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A}))) \preceq_{\mathsf{list2}} \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})), \\ \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle \\ \text{———————— Equation 3 ————————} \\ \left\langle \begin{array}{c} \varphi, \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A}))) \preceq_{\mathsf{list2}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{right\_list}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle \end{array}$$

where in order to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{list2}\ \varphi \to (\Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true})$$

has to be proved.

The corresponding difference predicate, $\Delta^2_{\mathsf{delete}} : \mathsf{nat} \times \mathsf{list2} \to \mathsf{bool}$, is now synthesized with the simplified difference formulas from the derivations in the Estimation Calculus as:

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \Delta^2_{\mathsf{delete}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A}))\\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A}))\\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A})) \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge\\ \mathsf{member}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\\ \qquad \to \left(\begin{array}{l}\Delta^2_{\mathsf{delete}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\Delta^2_{\mathsf{delete}}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{app}}(\mathsf{delete}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})),\mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{false}\end{array}\right)\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A})) \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge\\ \mathsf{member}(\mathsf{x},\mathsf{left\_list}(\mathsf{A})) \equiv \mathsf{false}\end{array}\right)\\ \qquad \to \left(\begin{array}{l}\Delta^2_{\mathsf{delete}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\Delta^2_{\mathsf{delete}}(\mathsf{x},\mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}),\mathsf{delete}(\mathsf{x},\mathsf{right\_list}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\end{array}\right)\end{array}$$

6.5 last : list2 → nat

last computes the last element in a non-empty list, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \to \mathsf{last}(\mathsf{A}) \equiv \mathsf{get\_nat}(\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}{:}\,\mathsf{list2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A})) \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \mathsf{last}(\mathsf{A}) \equiv \mathsf{last}(\mathsf{right\_list}(\mathsf{A}))\end{array}$$

The recursion ordering of last is well-founded. There is only one recursive definition case with a single recursive call of last. Hence, we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A})) \wedge\\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_list(A)} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we need to show

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \land\\ \mathsf{A} \not\equiv \mathsf{nil} \land \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))}\end{array}\right)\\ \quad\to (\mathsf{false} \lor \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be easily done using the definitions of $\Delta^1_{\mathsf{right\_list}}$ and Γ_{app}.

6.6 butlast : list2 → list2

butlast computes the original list without its last element, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list2}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{butlast(A)} \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list2}\\ \quad \mathsf{A} \equiv \mathsf{single(get\_nat(A))} \to \mathsf{butlast(A)} \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{list2}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \land\\ \mathsf{A} \not\equiv \mathsf{nil} \land \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))}\end{array}\right)\\ \quad\to \mathsf{butlast(A)} \equiv \mathsf{app(left\_list(A), butlast(right\_list(A)))}\end{array}$$

The recursion ordering of butlast is well-founded. There is only one recursive definition case with a single recursive call of butlast. Hence, we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app(left\_list(A), right\_list(A))} \land\\ \mathsf{A} \not\equiv \mathsf{nil} \land \mathsf{A} \not\equiv \mathsf{single(get\_nat(A))}\end{array}\right)$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_list(A)} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we need to show

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{list2}\\ \quad\left(\begin{array}{c}\mathsf{A}\equiv\mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A}\not\equiv\mathsf{nil}\wedge\mathsf{A}\not\equiv\mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \quad\quad\rightarrow(\mathsf{false}\vee\Delta^1_{\mathsf{right\_list}}(\mathsf{A})\equiv\mathsf{true})\end{array}$$

which can be easily done using the definitions of $\Delta^1_{\mathsf{right\_list}}$ and Γ_{app}.

In addition, butlast denotes a 1-bounded function symbol. First of all butlast is completely specified, as proved by

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{list2}\\ \quad\mathsf{A}\equiv\mathsf{nil}\vee\\ \quad\mathsf{A}\equiv\mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\vee\\ \quad\left(\begin{array}{c}\mathsf{A}\equiv\mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A}\not\equiv\mathsf{nil}\wedge\mathsf{A}\not\equiv\mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right).\end{array}$$

Then, we examine each definition case of butlast separately. For the first definition case we abbreviate the invariant case condition

$$\mathsf{A}\equiv\mathsf{nil}$$

by φ, and we obtain

$$\dfrac{\dfrac{-}{\langle\varphi,\mathsf{nil}\preceq_{\mathsf{list2}}\mathsf{nil},\mathsf{false}\rangle}\ \text{Identity}}{\langle\varphi,\mathsf{nil}\preceq_{\mathsf{list2}}\mathsf{A},\mathsf{false}\rangle}\ \text{Equation 1}$$

For the second definition case we abbreviate the invariant case condition

$$\mathsf{A}\equiv\mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))$$

by φ, and we obtain

$$\dfrac{\dfrac{-}{\langle\varphi,\mathsf{nil}\preceq_{\mathsf{list2}}\mathsf{single}(\mathsf{get\_nat}(\mathsf{A})),\mathsf{false}\rangle}\ \text{Equivalence}}{\langle\varphi,\mathsf{nil}\preceq_{\mathsf{list2}}\mathsf{A},\mathsf{false}\rangle}\ \text{Equation 1}$$

For the third definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A}\equiv\mathsf{app}(\mathsf{left\_list}(\mathsf{A}),\mathsf{right\_list}(\mathsf{A}))\wedge\\ \mathsf{A}\not\equiv\mathsf{nil}\wedge\mathsf{A}\not\equiv\mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)$$

by φ, and since this is a recursive case, we may assume

$$\xi\Rightarrow\ \dfrac{\langle\varphi,\mathsf{right\_list}(\mathsf{A})\preceq_{\mathsf{list2}}\mathsf{A},\Delta\rangle}{\langle\varphi,\mathsf{butlast}(\mathsf{right\_list}(\mathsf{A}))\preceq_{\mathsf{list2}}\mathsf{right\_list}(\mathsf{A}),\Delta^1_{\mathsf{butlast}}(\mathsf{right\_list}(\mathsf{A}))\equiv\mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

as an additional inference rule, where ξ is an abbreviation for

$$\forall\, \mathsf{A}\!:\!\mathsf{list2}\ \varphi \to \Delta$$

Then, we obtain:

$$\cfrac{-}{\langle \varphi, \mathsf{left\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{left\_list}(\mathsf{A}), \mathsf{false}\rangle}\ \mathsf{Identity}$$

On the other hand, we can derive:

$$\cfrac{\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}}{\langle \varphi, \mathsf{butlast}(\mathsf{right\_list}(\mathsf{A})) \preceq_{\mathsf{list2}} \mathsf{right\_list}(\mathsf{A}), \Delta^2_{\mathsf{butlast}}(\mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \mathsf{Induction\ Hypothesis}$$

where in order to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}\!:\!\mathsf{list2}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_list}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved. Now, we can continue with the derivation:

$$\cfrac{\cfrac{\begin{array}{c}\langle \varphi, \mathsf{left\_list}(\mathsf{A}) \preceq_{\mathsf{list2}} \mathsf{left\_list}(\mathsf{A}), \mathsf{false}\rangle\ ,\\ \langle \varphi, \mathsf{butlast}(\mathsf{right\_list}(\mathsf{A})) \preceq_{\mathsf{list2}} \mathsf{right\_list}(\mathsf{A}), \Delta^2_{\mathsf{butlast}}(\mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}\rangle\end{array}}{\left\langle \begin{array}{c}\varphi, \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{butlast}(\mathsf{right\_list}(\mathsf{A}))) \preceq_{\mathsf{list2}} \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})),\\ \mathsf{false} \vee \Delta^2_{\mathsf{butlast}}(\mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{butlast}(\mathsf{right\_list}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right\rangle}\ \mathsf{Weak\ Embedding}}{\left\langle \begin{array}{c}\varphi, \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{butlast}(\mathsf{right\_list}(\mathsf{A}))) \preceq_{\mathsf{list2}} \mathsf{A},\\ \mathsf{false} \vee \Delta^2_{\mathsf{butlast}}(\mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{butlast}(\mathsf{right\_list}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right\rangle}\ \mathsf{Equation\ 3}$$

where to allow the application of the Weak Embedding Rule,

$$\forall\, \mathsf{A}\!:\!\mathsf{list2}\ \varphi \to \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true}$$

has to be proved.

The corresponding difference predicate, $\Delta^1_{\mathsf{butlast}} : \mathsf{list2} \to \mathsf{bool}$, is now synthesized with the simplified difference formulas from the derivations in the Estimation Calculus as:

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list2}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{list2}\\ \quad \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \to \Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{list2} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{app}(\mathsf{left\_list}(\mathsf{A}), \mathsf{right\_list}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right) \\ \quad \rightarrow \left(\begin{array}{c}\Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{true} \\ \leftrightarrow \left(\begin{array}{c}\Delta^1_{\mathsf{butlast}}(\mathsf{right\_list}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{app}}(\mathsf{left\_list}(\mathsf{A}), \mathsf{butlast}(\mathsf{right\_list}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\end{array}\right)\end{array}$$

6.7 $<_{\mathsf{list2}}$: list2 × list2 → bool

$<_{\mathsf{list2}}$ computes the less-than-relation on lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2} \\ \quad (\mathsf{A} <_{\mathsf{list2}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{length}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{length}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive constructive definition we are done. However note, that $<_{\mathsf{list2}}$ denotes a well-founded order relation.

6.8 $\leq_{\mathsf{list2}}$: list2 × list2 → bool

$\leq_{\mathsf{list2}}$ computes the less-than-or-equal-relation on lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2} \\ \quad (\mathsf{A} \leq_{\mathsf{list2}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{length}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{length}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive constructive definition we are done.

6.9 $>_{\mathsf{list2}}$: list2 × list2 → bool

$>_{\mathsf{list2}}$ computes the greater-than-relation on lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2} \\ \quad (\mathsf{A} >_{\mathsf{list2}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} <_{\mathsf{list2}} \mathsf{A}) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive constructive definition we are done.

6.10 $\geq_{\mathsf{list2}}$: list2 × list2 → bool

$\geq_{\mathsf{list2}}$ computes the greater-than-relation on lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{list2} \\ \quad (\mathsf{A} \geq_{\mathsf{list2}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} \leq_{\mathsf{list2}} \mathsf{A}) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive constructive definition we are done.

7 Error Lists, errorlist

This specification of error lists (of nats), $\mathsf{errorlist}$, uses three constructor functions $\mathsf{error} :\to \mathsf{errorlist}$, generating the error element, $\mathsf{nil} :\to \mathsf{errorlist}$, generating the empty list, and $\mathsf{cons} : \mathsf{nat} \times \mathsf{errorlist} \to \mathsf{errorlist}$, for the insertion of an element into an error list. Equality on $\mathsf{errorlist}$ is specified by the axioms:

$$\mathsf{error} \not\equiv \mathsf{nil}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{errorlist}\\ \quad \mathsf{error} \equiv \mathsf{cons}(\mathsf{x},\mathsf{A}) \leftrightarrow \mathsf{A} \equiv \mathsf{error}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{errorlist}\\ \quad \mathsf{nil} \not\equiv \mathsf{cons}(\mathsf{x},\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}{:}\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}{:}\mathsf{errorlist}\\ \quad \mathsf{cons}(\mathsf{x},\mathsf{A}) \equiv \mathsf{cons}(\mathsf{y},\mathsf{B})\\ \qquad \leftrightarrow \begin{pmatrix} (\mathsf{A} \equiv \mathsf{error} \wedge \mathsf{B} \equiv \mathsf{error}) \vee \\ (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{A} \equiv \mathsf{B}) \end{pmatrix}\end{array}$$

By the above specification we have defined a non-freely generated data type. Hence, we must prove the constructor function cons to be size increasing by using the respective implementation specification. Furthermore, the strictness predicate $\Theta^2_{\mathsf{cons}} : \mathsf{nat} \times \mathsf{errorlist} \to \mathsf{bool}$ and the minimal representation predicate $\Gamma_{\mathsf{cons}} : \mathsf{nat} \times \mathsf{errorlist} \to \mathsf{bool}$ have to be synthesized.

The implementation specification is automatically generated using the constructor functions $\mathsf{error}_{\mathrm{I}} :\to \mathsf{errorlist}_{\mathrm{I}}$, $\mathsf{nil}_{\mathrm{I}} :\to \mathsf{errorlist}_{\mathrm{I}}$, $\mathsf{cons}_{\mathrm{I}} : \mathsf{nat} \times \mathsf{errorlist}_{\mathrm{I}} \to \mathsf{errorlist}_{\mathrm{I}}$, and the new equality predicate $\mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}} : \mathsf{errorlist}_{\mathrm{I}} \times \mathsf{errorlist}_{\mathrm{I}} \to \mathsf{bool}$.

$$\mathsf{error}_{\mathrm{I}} \not\equiv \mathsf{nil}_{\mathrm{I}}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{error}_{\mathrm{I}} \not\equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{nil}_{\mathrm{I}} \not\equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}:\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{y},\mathsf{B}) \to (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{A} \equiv \mathsf{B})\end{array}$$

$$\mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{error}_{\mathrm{I}},\mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{false}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{error}_{\mathrm{I}},\mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A})) \equiv \mathsf{true} \leftrightarrow \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{A},\mathsf{error}_{\mathrm{I}}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{nil}_{\mathrm{I}},\mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A})) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}:\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A}),\mathsf{cons}_{\mathrm{I}}(\mathsf{y},\mathsf{B})) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}(\mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{A},\mathsf{error}_{\mathrm{I}}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{B},\mathsf{error}_{\mathrm{I}}) \equiv \mathsf{true}) \vee \\ (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true})\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{A},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \to \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{B},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B},\mathsf{C}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad (\mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{B},\mathsf{C}) \equiv \mathsf{true})\\ \qquad \to \mathsf{Eq}_{\mathsf{errorlist}_{\mathrm{I}}}(\mathsf{A},\mathsf{C}) \equiv \mathsf{true}\end{array}$$

Since $\mathsf{errorlist}_{\mathrm{I}}$ is freely generated, the strictness predicate $\theta^2_{\mathsf{cons}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{errorlist}_{\mathrm{I}} \to \mathsf{bool}$, as well as the minimal representation predicate $\gamma_{\mathsf{cons}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{errorlist}_{\mathrm{I}} \to \mathsf{bool}$ are defined by:

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \theta^2_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \gamma_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

In addition, all constructor functions of $\mathsf{errorlist}_{\mathrm{I}}$ are non-overlapping. Hence, for the constructor function $\mathsf{cons}_{\mathrm{I}}$ we introduce two destructor functions $\mathsf{car}_{\mathrm{I}} : \mathsf{errorlist}_{\mathrm{I}} \to \mathsf{nat}$ for the first argument of $\mathsf{cons}_{\mathrm{I}}$ and $\mathsf{cdr}_{\mathrm{I}} : \mathsf{errorlist}_{\mathrm{I}} \to \mathsf{errorlist}_{\mathrm{I}}$ for the second argument of $\mathsf{cons}_{\mathrm{I}}$. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}:\mathsf{errorlist}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{B}) \to \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{car}_{\mathrm{I}}(\mathsf{A}),\mathsf{cdr}_{\mathrm{I}}(\mathsf{A}))\end{array}$$

$$\mathsf{car}_{\mathrm{I}}(\mathsf{nil}_{\mathrm{I}}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\mathsf{car_I}(\mathsf{error_I}) \equiv 0 \ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\mathsf{cdr_I}(\mathsf{nil_I}) \equiv \mathsf{nil_I}$$

$$\mathsf{cdr_I}(\mathsf{error_I}) \equiv \mathsf{error_I}$$

$$\begin{array}{l}\forall\ \mathsf{x}\!:\!\mathsf{nat}\ \forall\ \mathsf{A},\mathsf{B}\!:\!\mathsf{errorlist_I}\\ \quad \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{x},\mathsf{B}) \rightarrow \gamma_{\mathsf{cons_I}}(\mathsf{car_I}(\mathsf{A}),\mathsf{cdr_I}(\mathsf{A})) \equiv \mathsf{true}\end{array}$$

Now, $\mathsf{cdr_I}$ is 1-bounded with difference predicate $\Delta^{\mathrm{I1}}_{\mathsf{cdr_I}} : \mathsf{errorlist_I} \rightarrow \mathsf{bool}$, defined by

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \Delta^{\mathrm{I1}}_{\mathsf{cdr_I}}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}),\mathsf{cdr_I}(\mathsf{A}))\end{array}$$

Furthermore, the function $\mathsf{term\_size}_{\mathsf{errorlist_I}} : \mathsf{errorlist_I} \rightarrow \mathsf{nat}$ is synthesized by:

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \mathsf{A} \equiv \mathsf{nil_I} \rightarrow \mathsf{term\_size}_{\mathsf{errorlist_I}}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \mathsf{A} \equiv \mathsf{error_I} \rightarrow \mathsf{term\_size}_{\mathsf{errorlist_I}}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}),\mathsf{cdr_I}(\mathsf{A}))\\ \qquad \rightarrow \mathsf{term\_size}_{\mathsf{errorlist_I}}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{term\_size}_{\mathsf{errorlist_I}}(\mathsf{cdr_I}(\mathsf{A})))\end{array}$$

In order to have easier proofs, we specify a function $\mathsf{min\_size}_{\mathsf{errorlist_I}} : \mathsf{errorlist_I} \rightarrow \mathsf{nat}$, by

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \mathsf{A} \equiv \mathsf{nil_I} \rightarrow \mathsf{min\_size}_{\mathsf{errorlist_I}}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \mathsf{A} \equiv \mathsf{error_I} \rightarrow \mathsf{min\_size}_{\mathsf{errorlist_I}}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}),\mathsf{cdr_I}(\mathsf{A})) \wedge\\ \mathsf{Eq}_{\mathsf{errorlist_I}}(\mathsf{A},\mathsf{error_I}) \equiv \mathsf{true}\end{array}\right)\\ \qquad \rightarrow \mathsf{min\_size}_{\mathsf{errorlist_I}}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}),\mathsf{cdr_I}(\mathsf{A})) \wedge\\ \mathsf{Eq}_{\mathsf{errorlist_I}}(\mathsf{A},\mathsf{error_I}) \equiv \mathsf{false}\end{array}\right)\\ \qquad \rightarrow \mathsf{min\_size}_{\mathsf{errorlist_I}}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{errorlist_I}}(\mathsf{cdr_I}(\mathsf{A})))\end{array}$$

The specification of $\mathsf{min\_size}_{\mathsf{errorlist_I}}$ is case-distinct, as proved by

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \neg\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{nil_I} \wedge\\ \mathsf{A} \equiv \mathsf{error_I}\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist_I}\\ \quad \neg\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{nil_I} \wedge\\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}),\mathsf{cdr_I}(\mathsf{A})) \wedge\\ \mathsf{Eq}_{\mathsf{errorlist_I}}(\mathsf{A},\mathsf{error_I}) \equiv \mathsf{true}\end{array}\right)\end{array}\right)\end{array}$$

$$\forall\, \mathsf{A}\colon \mathsf{errorlist_I}\quad \neg\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{nil_I} \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})) \wedge \\ \mathsf{Eq_{errorlist_I}}(\mathsf{A}, \mathsf{error_I}) \equiv \mathsf{false} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{A}\colon \mathsf{errorlist_I}\quad \neg\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{error_I} \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})) \wedge \\ \mathsf{Eq_{errorlist_I}}(\mathsf{A}, \mathsf{error_I}) \equiv \mathsf{true} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{A}\colon \mathsf{errorlist_I}\quad \neg\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{error_I} \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})) \wedge \\ \mathsf{Eq_{errorlist_I}}(\mathsf{A}, \mathsf{error_I}) \equiv \mathsf{false} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{A}\colon \mathsf{errorlist_I}\quad \neg\left(\begin{array}{c} \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})) \wedge \\ \mathsf{Eq_{errorlist_I}}(\mathsf{A}, \mathsf{error_I}) \equiv \mathsf{true} \end{array}\right) \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})) \wedge \\ \mathsf{Eq_{errorlist_I}}(\mathsf{A}, \mathsf{error_I}) \equiv \mathsf{false} \end{array}\right) \end{array}\right)$$

Furthermore, the recursion ordering of $\mathsf{min\_size_{errorlist_I}}$ is well-founded. To prove that we use the Estimation Calculus. There is only one recursive case with a single recursive call of $\mathsf{min\_size_{errorlist_I}}$. Now, we abbreviate the case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})) \wedge \\ \mathsf{Eq_{errorlist_I}}(\mathsf{A}, \mathsf{error_I}) \equiv \mathsf{false} \end{array}\right)$$

by φ. Then, using the Estimation Calculus, we obtain:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist_I}} \mathsf{A}, \mathsf{false}\rangle}\;\mathsf{Identity}}{\left\langle \varphi, \mathsf{cdr_I}(\mathsf{A}) \preceq_{\mathsf{errorlist_I}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{cdr_I}}(\mathsf{A})\right\rangle}\;\mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\forall\, \mathsf{A}\colon \mathsf{errorlist_I}\quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})) \wedge \\ \mathsf{Eq_{errorlist_I}}(\mathsf{A}, \mathsf{error_I}) \equiv \mathsf{false} \end{array}\right) \to (\mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{cdr_I}}(\mathsf{A}))$$

which can be simplified to

$$\forall\, \mathsf{A}\colon \mathsf{errorlist_I}\quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})) \wedge \\ \mathsf{Eq_{errorlist_I}}(\mathsf{A}, \mathsf{error_I}) \equiv \mathsf{false} \end{array}\right) \to \mathsf{A} \equiv \mathsf{cons_I}(\mathsf{car_I}(\mathsf{A}), \mathsf{cdr_I}(\mathsf{A})).$$

Now, we need to prove that the above axiomatization of $\mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}$ computes the minimal size of an error list, indeed. Therefore we need to show the following proof obligations

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}\!:\!\mathsf{errorlist}_\mathrm{I}\\ \quad \mathsf{Eq}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \rightarrow (\mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist}_\mathrm{I}\ \exists\ \mathsf{B}\!:\!\mathsf{errorlist}_\mathrm{I}\\ \quad \mathsf{Eq}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \wedge (\mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A}) \geq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}\!:\!\mathsf{errorlist}_\mathrm{I}\\ \quad \mathsf{Eq}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \rightarrow \mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A}) \equiv \mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{B})\end{array}$$

Next, we need to show that cons denotes a size increasing constructor function. To do that, we prove:

$$\begin{array}{l}\forall\ \mathsf{x}\!:\!\mathsf{nat}\ \forall\ \mathsf{A}\!:\!\mathsf{errorlist}_\mathrm{I}\\ \quad (\mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{cons}_\mathrm{I}(\mathsf{x},\mathsf{A}))) \equiv \mathsf{true}.\end{array}$$

Finally, we need to define the strictness predicate $\Theta^2_{\mathsf{cons}_\mathrm{I}} : \mathsf{nat} \times \mathsf{errorlist}_\mathrm{I} \rightarrow \mathsf{bool}$ and the minimal representation predicate $\Gamma_{\mathsf{cons}_\mathrm{I}} : \mathsf{nat} \times \mathsf{errorlist}_\mathrm{I} \rightarrow \mathsf{bool}$. We suggest the following definitions:

$$\begin{array}{l}\forall\ \mathsf{x}\!:\!\mathsf{nat}\ \forall\ \mathsf{A}\!:\!\mathsf{errorlist}_\mathrm{I}\\ \quad \Theta^2_{\mathsf{cons}_\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{Eq}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A},\mathsf{error}_\mathrm{I}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{x}\!:\!\mathsf{nat}\ \forall\ \mathsf{A}\!:\!\mathsf{errorlist}_\mathrm{I}\\ \quad \Gamma_{\mathsf{cons}_\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{Eq}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A},\mathsf{error}_\mathrm{I}) \equiv \mathsf{false}\end{array}$$

However, we have to prove that our suggestions really define the strictness and the minimal representation predicate. Hence, we need to show that

$$\begin{array}{l}\forall\ \mathsf{x}\!:\!\mathsf{nat}\ \forall\ \mathsf{A}\!:\!\mathsf{errorlist}_\mathrm{I}\\ \quad \Theta^2_{\mathsf{cons}_\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow (\mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{cons}_\mathrm{I}(\mathsf{x},\mathsf{A}))) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{x}\!:\!\mathsf{nat}\ \forall\ \mathsf{A}\!:\!\mathsf{errorlist}_\mathrm{I}\\ \quad \Gamma_{\mathsf{cons}_\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{cons}_\mathrm{I}(\mathsf{x},\mathsf{A})) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{errorlist}_\mathrm{I}}(\mathsf{A}))\end{array}$$

Having done so, we know for our original specification errorlist that the constructor function cons is size increasing, and we can translate the strictness predicate as well as the minimal representation predicate into the original specification. Hence, we obtain:

$$\begin{array}{l}\forall\ \mathsf{x}\!:\!\mathsf{nat}\ \forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \Theta^2_{\mathsf{cons}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{A} \not\equiv \mathsf{error}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{x}\!:\!\mathsf{nat}\ \forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \Gamma_{\mathsf{cons}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{A} \not\equiv \mathsf{error}\end{array}$$

The data type errorlist possesses overlapping constructor functions, since

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\\ \quad \mathsf{error} \equiv \mathsf{cons}(\mathsf{x}, \mathsf{error})\end{array}$$

Thus, we cannot use the simplified construction scheme for the destructor functions.

For the constructor function cons we introduce two destructor functions $\mathsf{car} : \mathsf{errorlist} \to \mathsf{nat}$ for the first argument of cons and $\mathsf{cdr} : \mathsf{errorlist} \to \mathsf{errorlist}$ for the second argument of cons. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{x}, \mathsf{B}) \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{B}))\end{array}$$

$$\mathsf{car}(\mathsf{nil}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\mathsf{car}(\mathsf{error}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\mathsf{cdr}(\mathsf{nil}) \equiv \mathsf{nil}$$

$$\mathsf{cdr}(\mathsf{error}) \equiv \mathsf{nil}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{x}, \mathsf{B}) \wedge \mathsf{A} \not\equiv \mathsf{error})\\ \qquad \to \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{x}, \mathsf{B}) \wedge \mathsf{A} \equiv \mathsf{error})\\ \qquad \to \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false}\end{array}$$

The reflexive destructor function of the constructor function cons, cdr, is 1-bounded, and the difference predicate $\Delta^1_{\mathsf{cdr}} : \mathsf{errorlist} \to \mathsf{bool}$ is defined by

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge\\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

For the data type errorlist we will give constructive function and predicate specifications for app, member, min, max, length, delete, last, butlast, sort, $<_{\mathsf{errorlist}}$, $\leq_{\mathsf{errorlist}}$, $>_{\mathsf{errorlist}}$, and $\geq_{\mathsf{errorlist}}$.

7.1 app : errorlist × errorlist → errorlist

app computes the concatenation of two error lists and is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{error} \to \mathsf{app}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{error}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{app}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{B}\end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}) \\ \qquad \to \mathsf{app}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{app}(\mathsf{cdr}(\mathsf{A}), \mathsf{B})) \end{array}$$

The recursion ordering of app is well-founded. There is only one definition case with a single recursive call of app. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error})$$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{list}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{list} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}) \\ \qquad \to \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \end{array} \right). \end{array}$$

This formula can be easily proved using the definition of the destructor functions.

7.2 member : nat × errorlist → bool

member computes the containment relation of an element in an error list and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{errorlist} \\ \quad \mathsf{A} \equiv \mathsf{error} \to \mathsf{member}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{errorlist} \\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{member}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \equiv \mathsf{car}(\mathsf{A})) \\ \qquad \to \mathsf{member}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A})) \\ \qquad \to \mathsf{member}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{member}(\mathsf{x}, \mathsf{cdr}(\mathsf{A})) \end{array}$$

The recursion ordering of member is well-founded. There is only one definition case with a single recursive call of member. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))$$

by φ, we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A})) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A})) \\ \qquad \to \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right) \end{array}$$

This formula can be easily proved using the definition of the destructor functions.

7.3 length : errorlist → nat

length computes the length of an error list and is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}{:}\,\mathsf{errorlist} \\ \quad \mathsf{A} \equiv \mathsf{error} \to \mathsf{length}(\mathsf{A}) \equiv 0 \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}{:}\,\mathsf{errorlist} \\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{length}(\mathsf{A}) \equiv 0 \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}{:}\,\mathsf{errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}) \\ \qquad \to \mathsf{length}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{length}(\mathsf{cdr}(\mathsf{A}))) \end{array}$$

The recursion ordering of length is well-founded. There is only one definition case with a single recursive call of length. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error})$$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))\\ \qquad \to \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge\\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

This formula can be easily proved using the definition of the destructor functions.

7.4 delete : nat × errorlist → errorlist

delete computes the delete operation on error lists, thus it removes the first occurrence of a specified object in an error list, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{error} \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \equiv \mathsf{car}(\mathsf{A}))\\ \qquad \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{cdr}(\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))\\ \qquad \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A})))\end{array}$$

The recursion ordering of delete is well-founded. There is only one definition case with a single recursive call of member. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))$$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))\\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))\\ \qquad \rightarrow \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

This formula can be easily proved using the definition of the destructor functions.

In addition, delete denotes a 2-bounded function symbol. To prove this property, first of all, we need to show that delete is completely specified, i.e.,

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{error}\vee\\ \quad \mathsf{A} \equiv \mathsf{nil}\vee\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \equiv \mathsf{car}(\mathsf{A}))\vee\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))\end{array}$$

Then, we examine each definition case separately. For the first case we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \mathsf{A} \equiv \mathsf{nil}, \mathsf{error} \preceq_{\mathsf{errorlist}} \mathsf{error}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \mathsf{A} \equiv \mathsf{error}, \mathsf{error} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the second case we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \mathsf{A} \equiv \mathsf{nil}, \mathsf{nil} \preceq_{\mathsf{errorlist}} \mathsf{nil}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \mathsf{A} \equiv \mathsf{nil}, \mathsf{nil} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the third case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \equiv \mathsf{car}(\mathsf{A}))$$

by φ, and we obtain the derivation in the Estimation Calculus:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

And, for the fourth case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car}(\mathsf{A}))$$

by φ. Furthermore, since this case is recursive, we can assume an additional inference rule as the induction hypothesis:

$$\xi \Rightarrow \quad \cfrac{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \Delta\rangle}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where ξ is an abbreviation for the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{errorlist}\ \varphi \rightarrow \Delta$$

Then, we obtain the derivation:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{errorlist}} \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})), \\ \Delta^2_{\mathsf{delete}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{errorlist}} \mathsf{A}, \\ \Delta^2_{\mathsf{delete}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{errorlist}\ \varphi \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and in order to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{x{:}nat}\ \forall\, \mathsf{A{:}errorlist}\ \varphi \rightarrow \Gamma_{\mathsf{cons}}(\mathsf{car(A)}, \mathsf{cdr(A)}) \equiv \mathsf{true}$$

needs to be shown.

To synthesize the difference predicate $\Delta^2_{\mathsf{delete}} : \mathsf{nat} \times \mathsf{errorlist} \rightarrow \mathsf{bool}$, we use the simplified difference formulas from each derivation, and we obtain:

$$\begin{array}{l} \forall\, \mathsf{x{:}nat}\ \forall\, \mathsf{A{:}errorlist} \\ \quad \mathsf{A} \equiv \mathsf{error} \rightarrow \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x{:}nat}\ \forall\, \mathsf{A{:}errorlist} \\ \quad \mathsf{A} \equiv \mathsf{nil} \rightarrow \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x{:}nat}\ \forall\, \mathsf{A{:}errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \equiv \mathsf{car(A)}) \\ \qquad \rightarrow \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x{:}nat}\ \forall\, \mathsf{A{:}errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{x} \not\equiv \mathsf{car(A)}) \\ \qquad \rightarrow \left(\begin{array}{l} \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true} \\ \qquad \leftrightarrow \left(\begin{array}{c} \Delta^2_{\mathsf{delete}}(\mathsf{cdr(A)}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{cons}}(\mathsf{car(A)}, \mathsf{delete(x, cdr(A))}) \equiv \mathsf{false} \end{array} \right) \end{array} \right) \end{array}$$

7.5 min : errorlist → nat

min computes the minimal element in a non-empty error list, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A{:}errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{cdr(A)} \equiv \mathsf{nil}) \\ \qquad \rightarrow \mathsf{min(A)} \equiv \mathsf{car(A)} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A{:}errorlist} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)), cdr(cdr(A)))} \wedge \\ (\mathsf{car(A)} <_{\mathsf{nat}} \mathsf{car(cdr(A))}) \equiv \mathsf{true} \end{array} \right) \\ \qquad \rightarrow \mathsf{min(A)} \equiv \mathsf{min(cons(car(A), cdr(cdr(A))))} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A{:}errorlist} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)), cdr(cdr(A)))} \wedge \\ (\mathsf{car(A)} <_{\mathsf{nat}} \mathsf{car(cdr(A))}) \equiv \mathsf{false} \end{array} \right) \\ \qquad \rightarrow \mathsf{min(A)} \equiv \mathsf{min(cdr(A))} \end{array}$$

The recursion ordering of min is well-founded. There are two definition cases with one recursive call in each. For the first recursive case we obtain the derivation in the Estimation Calculus, abbreviating the case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)), cdr(cdr(A)))} \wedge \\ (\mathsf{car(A)} <_{\mathsf{nat}} \mathsf{car(cdr(A))}) \equiv \mathsf{true} \end{array} \right)$$

by φ:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{cdr(A)} \preceq_{\mathsf{errorlist}} \mathsf{cdr(A)}, \mathsf{false}\rangle}\text{ Identity}}{\left\langle \begin{array}{c}\varphi, \mathsf{cdr(cdr(A))} \preceq_{\mathsf{errorlist}} \mathsf{cdr(A)},\\ \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{cdr(A)}) \equiv \mathsf{true}\end{array}\right\rangle}\text{ Estimation}}{\left\langle \begin{array}{c}\varphi, \mathsf{cons(car(A), cdr(cdr(A)))} \preceq_{\mathsf{errorlist}} \mathsf{cons(car(A), cdr(A))},\\ \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{cdr(A)}) \equiv \mathsf{true} \lor \Gamma_{\mathsf{cons}}(\mathsf{car(A), cdr(cdr(A))}) \equiv \mathsf{false}\end{array}\right\rangle}\text{ Weak Embedding}}{\left\langle \begin{array}{c}\varphi, \mathsf{cons(car(A), cdr(cdr(A)))} \preceq_{\mathsf{errorlist}} \mathsf{A},\\ \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{cdr(A)}) \equiv \mathsf{true} \lor \Gamma_{\mathsf{cons}}(\mathsf{car(A), cdr(cdr(A))}) \equiv \mathsf{false}\end{array}\right\rangle}\text{ Equation 3}$$

where to enable the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}\!:\!\mathsf{errorlist}\ \varphi \to \Gamma_{\mathsf{cons}}(\mathsf{car(A), cdr(A)}) \equiv \mathsf{true}$$

has to be shown. In order to ensure the strict relation, we, therefore, need to prove

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{errorlist}\\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \land \mathsf{A} \not\equiv \mathsf{error} \land\\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)), cdr(cdr(A)))} \land\\ (\mathsf{car(A)} <_{\mathsf{nat}} \mathsf{car(cdr(A))}) \equiv \mathsf{true}\end{array}\right)\\ \to \left(\begin{array}{c}\mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{cdr(A)}) \equiv \mathsf{true} \lor\\ \Gamma_{\mathsf{cons}}(\mathsf{car(A), cdr(cdr(A))}) \equiv \mathsf{false}\end{array}\right)\end{array}$$

which can be easily proved using the definitions of the involved functions. For the second recursive case we abbreviate the case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \land \mathsf{A} \not\equiv \mathsf{error} \land\\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)), cdr(cdr(A)))} \land\\ (\mathsf{car(A)} <_{\mathsf{nat}} \mathsf{car(cdr(A))}) \equiv \mathsf{false}\end{array}\right)$$

by φ, and we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\text{ Identity}}{\langle \varphi, \mathsf{cdr(A)} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\text{ Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{errorlist}\\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \land \mathsf{A} \not\equiv \mathsf{error} \land\\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)), cdr(cdr(A)))} \land\\ (\mathsf{car(A)} <_{\mathsf{nat}} \mathsf{car(cdr(A))}) \equiv \mathsf{false}\end{array}\right)\\ \to (\mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be easily proved using the definition of the involved functions.

7.6 max : errorlist $\rightarrow$ nat

max computes the maximal element in a non-empty error list, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error} \wedge \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil}) \\ \quad\quad \rightarrow \mathsf{max}(\mathsf{A}) \equiv \mathsf{car}(\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right) \\ \quad\quad \rightarrow \mathsf{max}(\mathsf{A}) \equiv \mathsf{max}(\mathsf{cdr}(\mathsf{A}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right) \\ \quad\quad \rightarrow \mathsf{max}(\mathsf{A}) \equiv \mathsf{max}(\mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))))\end{array}$$

The recursion ordering of max is well-founded. There are two definition cases with one recursive call in each. For the first recursive case we obtain the derivation in the Estimation Calculus, abbreviating the case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right)$$

by φ:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\,\mathsf{Identity}}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\,\mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right) \\ \quad\quad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be easily proved using the definition of the involved functions.

For the second recursive case we abbreviate the case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\wedge \\ (\mathsf{car}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{car}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)$$

by φ, and we obtain the derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{cdr(A)} \preceq_{\mathsf{errorlist}} \mathsf{cdr(A)}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \begin{array}{c} \varphi, \mathsf{cdr(cdr(A))} \preceq_{\mathsf{errorlist}} \mathsf{cdr(A)}, \\ \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{cdr(A)}) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Estimation}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons(car(A), cdr(cdr(A)))} \preceq_{\mathsf{errorlist}} \mathsf{cons(car(A), cdr(A))}, \\ \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{cdr(A)}) \equiv \mathsf{true} \lor \Gamma_{\mathsf{cons}}(\mathsf{car(A), cdr(cdr(A))}) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons(car(A), cdr(cdr(A)))} \preceq_{\mathsf{errorlist}} \mathsf{A}, \\ \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{cdr(A)}) \equiv \mathsf{true} \lor \Gamma_{\mathsf{cons}}(\mathsf{car(A), cdr(cdr(A))}) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where to enable the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A{:}errorlist}\ \varphi \to \Gamma_{\mathsf{cons}}(\mathsf{car(A), cdr(A)}) \equiv \mathsf{true}$$

has to be shown. In order to ensure the strict relation, we, therefore, need to prove

$$\begin{array}{l} \forall\, \mathsf{A{:}errorlist} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \land \mathsf{A} \not\equiv \mathsf{error} \land \\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)), cdr(cdr(A)))} \land \\ (\mathsf{car(A)} <_{\mathsf{nat}} \mathsf{car(cdr(A))}) \equiv \mathsf{false} \end{array} \right) \\ \quad \to \left(\begin{array}{c} \mathsf{false} \lor \Delta^1_{\mathsf{cdr}}(\mathsf{cdr(A)}) \equiv \mathsf{true} \lor \\ \Gamma_{\mathsf{cons}}(\mathsf{car(A), cdr(cdr(A))}) \equiv \mathsf{false} \end{array} \right) \end{array}$$

which can be easily proved using the definitions of the involved functions.

7.7 last : errorlist → nat

last computes the last element in a non-empty error list, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A{:}errorlist} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \land \\ \mathsf{A} \not\equiv \mathsf{error} \land \\ \mathsf{cdr(A)} \equiv \mathsf{nil} \end{array} \right) \\ \quad \to \mathsf{last(A)} \equiv \mathsf{car(A)} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A{:}errorlist} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons(car(A), cdr(A))} \land \\ \mathsf{A} \not\equiv \mathsf{error} \\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)), cdr(cdr(A)))} \land \end{array} \right) \\ \quad \to \mathsf{last(A)} \equiv \mathsf{last(cdr(A))} \end{array}$$

The recursion ordering of last is well-founded. There is only one definition case with a single recursive call. Hence, we use the Estimation Calculus, abbreviating the case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \end{pmatrix}$$

by φ. We obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\big\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\big\rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \end{pmatrix} \\ \quad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be easily proved using the definitions of the involved functions.

7.8 butlast : errorlist → errorlist

butlast computes the original error list without its last element, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad \mathsf{A} \equiv \mathsf{error} \rightarrow \mathsf{butlast}(\mathsf{A}) \equiv \mathsf{error} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad \mathsf{A} \equiv \mathsf{nil} \rightarrow \mathsf{butlast}(\mathsf{A}) \equiv \mathsf{nil} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil} \end{pmatrix} \\ \quad \rightarrow \mathsf{butlast}(\mathsf{A}) \equiv \mathsf{nil} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{errorlist} \\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \end{pmatrix} \\ \quad \rightarrow \mathsf{butlast}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \end{array}$$

The recursion ordering of butlast is well-founded. There is only one definition case with a single recursive call. Hence, we use the Estimation Calculus, abbreviating the case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \end{pmatrix}$$

by φ. We obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{cdr(A)} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{errorlist}\\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons(car(A),cdr(A))}\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)),cdr(cdr(A)))}\end{array}\right)\\ \quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be easily prove using the definitions of the involved functions.

To prove that butlast is 1-bounded, first of all, we need to show that butlast is completely specified, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{error}\vee\\ \quad \mathsf{A} \equiv \mathsf{nil}\vee\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons(car(A),cdr(A))}\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ \mathsf{cdr(A)} \equiv \mathsf{nil}\end{array}\right)\vee\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons(car(A),cdr(A))}\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ \mathsf{cdr(A)} \equiv \mathsf{cons(car(cdr(A)),cdr(cdr(A)))}\end{array}\right)\end{array}$$

Then, we examine each definition case separately. For the first definition case we abbreviate the case condition

$$\mathsf{A} \equiv \mathsf{error}$$

by φ. Then, we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{error} \preceq_{\mathsf{errorlist}} \mathsf{error}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{error} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Equation\ 1}$$

For the second definition case we abbreviate the case condition

$$\mathsf{A} \equiv \mathsf{nil}$$

by φ. Then, we obtain the derivation:

$$\begin{array}{c} - \\ \text{——— Identity ———} \\ \langle \varphi, \mathsf{nil} \preceq_{\mathsf{errorlist}} \mathsf{nil}, \mathsf{false}\rangle \\ \text{——— Equation 1 ———} \\ \langle \varphi, \mathsf{nil} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle \end{array}$$

For the third definition case we abbreviate the case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil} \end{array}\right)$$

by φ, and we obtain the derivation:

$$\begin{array}{c} - \\ \text{——— Minimum ———} \\ \langle \varphi, \mathsf{nil} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{A} \not\equiv \mathsf{nil} \wedge \mathsf{A} \not\equiv \mathsf{error}\rangle \end{array}$$

For the fourth definition case we abbreviate the case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})), \mathsf{cdr}(\mathsf{cdr}(\mathsf{A}))) \end{array}\right)$$

by φ. Since this is a recursive definition case, we may assume the additional inference rule

$$\xi \Rightarrow \begin{array}{c} \langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \Delta\rangle \\ \text{——— Induction Hypothesis ———} \\ \langle \varphi, \mathsf{butlast}(\mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{A}), \Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\rangle \end{array}$$

as an induction hypothesis, where ξ is an abbreviation for the formula

$\forall\, \mathsf{A} : \mathsf{errorlist}\ \varphi \rightarrow \Delta$

Now, we obtain the derivation:

$$\begin{array}{c} - \\ \text{——— Identity ———} \\ \langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle \\ \text{——— Estimation ———} \\ \langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\rangle \\ \text{——— Induction Hypothesis ———} \\ \langle \varphi, \mathsf{butlast}(\mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{A}), \Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\rangle \\ \text{——— Weak Embedding ———} \\ \left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{errorlist}} \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})), \\ \Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle \\ \text{——— Equation 3 ———} \\ \left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \preceq_{\mathsf{errorlist}} \mathsf{A}, \\ \Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}), \mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle \end{array}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\ \varphi \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and to allow the application of the Weak Embedding Rule, the formula

$$\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}$$

needs to be shown.

Using the simplified difference formulas, we can now synthesize the definition of $\Delta^1_{\mathsf{butlast}}$: errorlist $\rightarrow$ bool:

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{error} \rightarrow \Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{nil} \rightarrow \Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{nil}\end{array}\right)\\ \quad\quad \rightarrow \Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ \mathsf{cdr}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{car}(\mathsf{cdr}(\mathsf{A})),\mathsf{cdr}(\mathsf{cdr}(\mathsf{A})))\end{array}\right)\\ \quad\quad \rightarrow \left(\begin{array}{l}\Delta^1_{\mathsf{butlast}}(\mathsf{A}) \equiv \mathsf{true}\\ \quad\quad \leftrightarrow \left(\begin{array}{c}\Delta^1_{\mathsf{butlast}}(\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{true}\vee\\ \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}),\mathsf{butlast}(\mathsf{cdr}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\end{array}\right)\end{array}$$

7.9 sort : errorlist $\rightarrow$ errorlist

sort sorts an error list, defined by:

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{error} \rightarrow \mathsf{sort}(\mathsf{A}) \equiv \mathsf{error}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad \mathsf{A} \equiv \mathsf{nil} \rightarrow \mathsf{sort}(\mathsf{A}) \equiv \mathsf{nil}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}\!:\!\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error})\\ \quad\quad \rightarrow \mathsf{sort}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{min}(\mathsf{A}),\mathsf{sort}(\mathsf{delete}(\mathsf{min}(\mathsf{A}),\mathsf{A})))\end{array}$$

The recursion ordering of sort is well-founded. There is only one recursive definition case with a single recursive call. Hence, we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error})$$

by φ. Using the Estimation Calculus, we obtain the following derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{delete}(\mathsf{min}(\mathsf{A}),\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{min}(\mathsf{A}),\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{errorlist}\\ \quad (\mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \wedge \mathsf{A} \not\equiv \mathsf{error})\\ \qquad \to (\mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{min}(\mathsf{A}),\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be proved by induction.

7.10 $<_{\mathsf{errorlist}}$: errorlist $\times$ errorlist $\to$ bool

$<_{\mathsf{errorlist}}$ computes the less-than-relation on error lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}{:}\mathsf{errorlist}\\ \quad \mathsf{B} \equiv \mathsf{error} \to (\mathsf{A} <_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}{:}\mathsf{errorlist}\\ \quad (\mathsf{B} \not\equiv \mathsf{error} \wedge \mathsf{A} \equiv \mathsf{error}) \to (\mathsf{A} <_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}{:}\mathsf{errorlist}\\ \quad \mathsf{B} \equiv \mathsf{nil} \to (\mathsf{A} <_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}{:}\mathsf{errorlist}\\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge\\ \mathsf{B} \not\equiv \mathsf{error}\wedge\\ \mathsf{A} \equiv \mathsf{nil}\end{array}\right)\\ \qquad \to (\mathsf{A} <_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}{:}\mathsf{errorlist}\\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge\\ \mathsf{B} \not\equiv \mathsf{error}\wedge\\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right)\\ \qquad \to (\mathsf{A} <_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}{:}\mathsf{errorlist}\\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge\\ \mathsf{B} \not\equiv \mathsf{error}\wedge\\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right)\\ \qquad \to (\mathsf{A} <_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{false}\end{array}$$

The recursion ordering of $<_{\mathsf{errorlist}}$ is well-founded: There are two definition cases with a single recursive call of $<_{\mathsf{errorlist}}$ in each. For each recursive definition case and each argument we use the Estimation Calculus. Starting with the first recursive case, we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \\ \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{pmatrix}$$

by φ. For the first argument of $<_{\mathsf{errorlist}}$, A, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true} \right\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{errorlist}}$-relation, we have to show

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist} \\ \quad \begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \\ \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{pmatrix} \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}), \end{array}$$

which can be done using the definitions of the involved functions. And for the second argument of $<_{\mathsf{errorlist}}$, B, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{B} \preceq_{\mathsf{errorlist}} \mathsf{B}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{cdr}(\mathsf{B}) \preceq_{\mathsf{errorlist}} \mathsf{B}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{B}) \equiv \mathsf{true} \right\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{errorlist}}$-relation, we have to show

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist} \\ \quad \begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \\ \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{pmatrix} \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{B}) \equiv \mathsf{true}), \end{array}$$

which can be done using the definitions of the involved functions.

For the second recursive definition case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{pmatrix}$$

by φ. For the first argument of $<_{\mathsf{errorlist}}$, A, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false}\rangle}\,\mathsf{Identity}}{\left\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\,\mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{errorlist}}$-relation, we have to show

$$\begin{array}{l} \forall\ \mathsf{A},\mathsf{B}:\mathsf{errorlist} \\ \quad \begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{pmatrix} \\ \quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{A}) \equiv \mathsf{true}), \end{array}$$

which can be done using the definitions of the involved functions. And for the second argument of $<_{\mathsf{errorlist}}$, B, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{B} \preceq_{\mathsf{errorlist}} \mathsf{B}, \mathsf{false}\rangle}\,\mathsf{Identity}}{\left\langle \varphi, \mathsf{cdr}(\mathsf{B}) \preceq_{\mathsf{errorlist}} \mathsf{B}, \mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{B}) \equiv \mathsf{true}\right\rangle}\,\mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{errorlist}}$-relation, we have to show

$$\begin{array}{l} \forall\ \mathsf{A},\mathsf{B}:\mathsf{errorlist} \\ \quad \begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{pmatrix} \\ \quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{cdr}}(\mathsf{B}) \equiv \mathsf{true}), \end{array}$$

which can be done using the definitions of the involved functions. Thus, the recursion ordering of $<_{\mathsf{errorlist}}$ is a well-founded ordering.

In addition, $<_{\mathsf{errorlist}}$ denotes a well-founded ordering as well. To prove that, we first have to show that $<_{\mathsf{errorlist}}$ is completely specified, i.e.,

$$\begin{array}{l}
\forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{errorlist} \\
\quad (\mathsf{B} \equiv \mathsf{error})\vee \\
\quad (\mathsf{B} \not\equiv \mathsf{error} \wedge \mathsf{A} \equiv \mathsf{error})\vee \\
\quad (\mathsf{B} \equiv \mathsf{nil})\vee \\
\quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{nil} \end{array}\right)\vee \\
\quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{array}\right)\vee \\
\quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{array}\right)
\end{array}$$

Next, for each definition case we show that

$$\forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{errorlist}\ (\mathsf{A} <_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{true} \rightarrow \mathsf{A} \prec_{\mathsf{errorlist}} \mathsf{B},$$

again, using the Estimation Calculus. For the first definition case we obtain

$$\frac{-}{\langle \mathsf{B} \equiv \mathsf{error} \wedge \mathsf{false} \equiv \mathsf{true}, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \Delta_1 \rangle}\ \mathsf{Tautology}$$

where in order to enable the application of the Tautology Rule, the first-order formula

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{errorlist} \\ \quad \neg(\mathsf{B} \equiv \mathsf{error} \wedge \mathsf{false} \equiv \mathsf{true}) \end{array}$$

has to be proved. To prove the strict relation, the formula

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{errorlist} \\ \quad (\mathsf{B} \equiv \mathsf{error} \wedge \mathsf{false} \equiv \mathsf{true}) \rightarrow \Delta_1 \end{array}$$

has to be shown. For the second definition case we obtain

$$\frac{-}{\left\langle \begin{array}{c} \mathsf{B} \not\equiv \mathsf{error} \wedge \mathsf{A} \equiv \mathsf{error} \wedge \mathsf{false} \equiv \mathsf{true}, \\ \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \Delta_2 \end{array} \right\rangle}\ \mathsf{Tautology}$$

where in order to enable the application of the Tautology Rule, the first-order formula

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{errorlist} \\ \quad \neg(\mathsf{B} \not\equiv \mathsf{error} \wedge \mathsf{A} \equiv \mathsf{error} \wedge \mathsf{false} \equiv \mathsf{true}) \end{array}$$

has to be proved. To prove the strict relation, the formula

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist} \\ \quad (\mathsf{B} \not\equiv \mathsf{error} \wedge \mathsf{A} \equiv \mathsf{error} \wedge \mathsf{false} \equiv \mathsf{true}) \rightarrow \Delta_2\end{array}$$

has to be shown. For the third definition case we obtain

$$\frac{-}{\langle \mathsf{B} \equiv \mathsf{nil} \wedge \mathsf{false} \equiv \mathsf{true}, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \Delta_3 \rangle}\ \text{Tautology}$$

where in order to enable the application of the Tautology Rule, the first-order formula

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist} \\ \quad \neg(\mathsf{B} \equiv \mathsf{nil} \wedge \mathsf{false} \equiv \mathsf{true})\end{array}$$

has to be proved. To prove the strict relation, the formula

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist} \\ \quad (\mathsf{B} \equiv \mathsf{nil} \wedge \mathsf{false} \equiv \mathsf{true}) \rightarrow \Delta_3\end{array}$$

has to be shown. For the fourth definition case we obtain the derivation

$$\cfrac{\cfrac{-}{\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{true} \equiv \mathsf{true}, \\ \mathsf{nil} \preceq_{\mathsf{errorlist}} \mathsf{B}, \mathsf{B} \not\equiv \mathsf{nil} \wedge \mathsf{B} \not\equiv \mathsf{error} \end{array} \right\rangle}\ \text{Minimum}}{\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{true} \equiv \mathsf{true}, \\ \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \mathsf{B} \not\equiv \mathsf{nil} \wedge \mathsf{B} \not\equiv \mathsf{error} \end{array} \right\rangle}\ \text{Equation 5}$$

showing the strict relation by

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist} \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{nil} \wedge \mathsf{true} \equiv \mathsf{true} \end{array} \right. \\ \quad\quad \left. \rightarrow (\mathsf{B} \not\equiv \mathsf{nil} \wedge \mathsf{B} \not\equiv \mathsf{error}) \right)\end{array}$$

The fifth definition case is a recursive case. Hence, we need to make an additional case analysis:

$(\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}$ or

$(\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}$.

For the first case we can assume as an induction hypothesis the inference rule:

$$\frac{-}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B}), \mathsf{true} \rangle}\ \text{Induction Hypothesis}$$

where we use φ as an abbreviation for

$$\left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge\\ \mathsf{B} \not\equiv \mathsf{error}\wedge\\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\wedge\\ \mathsf{true} \equiv \mathsf{true}\wedge\\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right)$$

Then, the derivation of

$$\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \Delta_4\rangle$$

is achieved by:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{cdr}(\mathsf{A}) \preceq_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B}), \mathsf{true}\rangle}\ \text{Induction Hypothesis}}{\left\langle \begin{array}{c}\varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{errorlist}} \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})),\\ \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false}\end{array}\right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c}\varphi, \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \preceq_{\mathsf{errorlist}} \mathsf{B},\\ \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false}\end{array}\right\rangle}\ \text{Equation 3}}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false}\rangle}\ \text{Equation 5}$$

where in order to enable the application of the Weak Embedding Rule, the first-order formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist}\ \varphi \rightarrow \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}$$

has to be shown. The strict relation is proved by

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist}\\ \quad \varphi \rightarrow (\mathsf{true} \vee \Gamma_{\mathsf{cons}}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A})) \equiv \mathsf{false}).\end{array}$$

For the second case we cannot assume an induction hypothesis. We prove the estimation formula

$$\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \Delta_5\rangle$$

where φ is an abbreviation for

$$\left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}),\mathsf{cdr}(\mathsf{B}))\wedge\\ \mathsf{B} \not\equiv \mathsf{error}\wedge\\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}),\mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\wedge\\ \mathsf{true} \equiv \mathsf{true}\wedge\\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right)$$

by an application of the Tautology Rule, where in order to enable this application, it is necessary to prove the first-order formula:

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist}\quad \neg\begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\wedge \\ \mathsf{true} \equiv \mathsf{true}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{pmatrix}$$

To prove the strict relation, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist}\quad \begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true}\wedge \\ \mathsf{true} \equiv \mathsf{true}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{pmatrix} \to \Delta_5$$

needs to be shown.

The sixth definition case is also a recursive case. Hence, we need to make an additional case analysis:

$$(\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \text{ or}$$

$$(\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}.$$

Although for the first case we could assume an induction hypothesis, this is not necessary since the derivation of the estimation formula

$$\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \Delta_6 \rangle$$

where φ is an abbreviation for

$$\begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B}))\wedge \\ \mathsf{B} \not\equiv \mathsf{error}\wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{error}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\wedge \\ \mathsf{false} \equiv \mathsf{true}\wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{pmatrix}$$

can be achieved by the application of the Tautology Rule. In order to enable this application, the first-order formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist}\quad \neg \begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \\ \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \wedge \\ \mathsf{false} \equiv \mathsf{true} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{pmatrix}$$

has to be shown. And for the strict relation, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist}\quad \begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \\ \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \wedge \\ \mathsf{false} \equiv \mathsf{true} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{true} \end{pmatrix} \to \Delta_6$$

needs to be proved. For the second case we cannot assume an induction hypothesis. We prove the estimation formula

$$\langle \varphi, \mathsf{A} \preceq_{\mathsf{errorlist}} \mathsf{B}, \Delta_7 \rangle$$

where φ is used as an abbreviation for

$$\begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \\ \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \wedge \\ \mathsf{false} \equiv \mathsf{true} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{pmatrix}$$

by an application of the Tautology Rule, where in order to enable this application, it is necessary to prove the first-order formula:

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{errorlist}\quad \neg \begin{pmatrix} \mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B})) \wedge \\ \mathsf{B} \not\equiv \mathsf{error} \wedge \\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{error} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \wedge \\ \mathsf{false} \equiv \mathsf{true} \wedge \\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false} \end{pmatrix}$$

To prove the strict relation, the formula

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\colon\mathsf{errorlist}\\ \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{B}), \mathsf{cdr}(\mathsf{B}))\wedge\\ \mathsf{B} \not\equiv \mathsf{error}\wedge\\ \mathsf{A} \equiv \mathsf{cons}(\mathsf{car}(\mathsf{A}), \mathsf{cdr}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{error}\wedge\\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\wedge\\ \mathsf{false} \equiv \mathsf{true}\wedge\\ (\mathsf{cdr}(\mathsf{A}) <_{\mathsf{errorlist}} \mathsf{cdr}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right)\\ \quad \to \Delta_7\end{array}$$

needs to be shown.

Having proved all these obligations, $<_{\mathsf{errorlist}}$ denotes a well-founded order relation.

7.11 $\leq_{\mathsf{errorlist}}$: errorlist × errorlist → bool

$\leq_{\mathsf{errorlist}}$ computes the less-than-or-equal-relation on error lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\colon\mathsf{errorlist}\\ \quad (\mathsf{A} \leq_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} <_{\mathsf{errorlist}} \mathsf{A}) \equiv \mathsf{false}\end{array}$$

Since this is a non-recursive constructive definition we are done.

7.12 $>_{\mathsf{errorlist}}$: errorlist × errorlist → bool

$>_{\mathsf{errorlist}}$ computes the greater-than-relation on error lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\colon\mathsf{errorlist}\\ \quad (\mathsf{A} >_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} <_{\mathsf{errorlist}} \mathsf{A}) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive constructive definition we are done.

7.13 $\geq_{\mathsf{errorlist}}$: errorlist × errorlist → bool

$\geq_{\mathsf{errorlist}}$ computes the greater-than-relation on error lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\colon\mathsf{errorlist}\\ \quad (\mathsf{A} \geq_{\mathsf{errorlist}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} \leq_{\mathsf{errorlist}} \mathsf{A}) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive constructive definition we are done.

8

Finite Sets, set

This specification of finite sets (of nats), set, uses two constructor functions empty $:\to$ set, generating the empty set, and ins : nat $\times$ set $\to$ set, for the insertion of an element into a set. Equality on set is specified using an auxiliary predicate $\in$: nat $\times$ set $\to$ bool and by the axioms:

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}$$
$$\mathsf{x} \notin \mathsf{empty}$$

$$\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set}$$
$$\mathsf{x} \in \mathsf{ins}(\mathsf{y},\mathsf{A}) \leftrightarrow (\mathsf{x} \equiv \mathsf{y} \vee \mathsf{x} \in \mathsf{A})$$

$$\forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{set}$$
$$\mathsf{A} \equiv \mathsf{B} \leftrightarrow (\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{x} \in \mathsf{A} \leftrightarrow \mathsf{x} \in \mathsf{B})$$

By the above specification we have defined a non-freely generated data type. Hence, we must prove the constructor function ins to be size increasing by using the respective implementation specification. Furthermore, the strictness predicate Θ^2_{ins} : nat $\times$ set $\to$ bool and the minimal representation predicate Γ_{ins} : nat $\times$ set $\to$ bool have to be synthesized.

The implementation specification is automatically generated using the constructor functions $\mathsf{empty}_\mathrm{I}$ $:\to$ set_I, ins_I : nat $\times$ set_I $\to$ set_I, as well as the new equality predicate $\mathsf{Eq}_{\mathsf{set}_\mathrm{I}}$: $\mathsf{set}_\mathrm{I} \times \mathsf{set}_\mathrm{I} \to$ bool.

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set}_\mathrm{I}$$
$$\mathsf{empty}_\mathrm{I} \not\equiv \mathsf{ins}_\mathrm{I}(\mathsf{x},\mathsf{A})$$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}_{\mathrm{I}}$
$\quad \mathsf{ins}_{\mathrm{I}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{y}, \mathsf{B}) \rightarrow (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{A} \equiv \mathsf{B})$

$\forall\, \mathsf{x} : \mathsf{nat}$
$\quad \mathsf{x} \notin_{\mathrm{I}} \mathsf{empty}_{\mathrm{I}}$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set}_{\mathrm{I}}$
$\quad \mathsf{x} \in \mathsf{ins}_{\mathrm{I}}(\mathsf{y}, \mathsf{A}) \leftrightarrow (\mathsf{x} \equiv \mathsf{y} \vee \mathsf{x} \in_{\mathrm{I}} \mathsf{A})$

$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$
$\qquad \leftrightarrow (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{x} \in_{\mathrm{I}} \mathsf{A} \leftrightarrow \mathsf{x} \in_{\mathrm{I}} \mathsf{B})$

$\forall\, \mathsf{A} : \mathsf{set}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{A}) \equiv \mathsf{true}$

$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \rightarrow \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{B}, \mathsf{A}) \equiv \mathsf{true}$

$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set}_{\mathrm{I}}$
$\quad (\mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{B}, \mathsf{C}) \equiv \mathsf{true})$
$\qquad \rightarrow \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{C}) \equiv \mathsf{true}$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}_{\mathrm{I}}$
$\quad (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true})$
$\qquad \rightarrow (\mathsf{x} \in_{\mathrm{I}} \mathsf{A} \leftrightarrow \mathsf{y} \in_{\mathrm{I}} \mathsf{B})$

Since $\mathsf{set}_{\mathrm{I}}$ is freely generated, the strictness predicate $\theta^2_{\mathsf{ins}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{set}_{\mathrm{I}} \rightarrow \mathsf{bool}$, as well as the minimal representation predicate $\gamma_{\mathsf{ins}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{set}_{\mathrm{I}} \rightarrow \mathsf{bool}$ are defined by:

$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set}_{\mathrm{I}}$
$\quad \theta^2_{\mathsf{ins}_{\mathrm{I}}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true}$

$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set}_{\mathrm{I}}$
$\quad \gamma_{\mathsf{ins}_{\mathrm{I}}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true}$

In addition, the constructor functions of $\mathsf{set}_{\mathrm{I}}$ are non-overlapping. Hence, for the constructor function $\mathsf{ins}_{\mathrm{I}}$ we introduce two destructor functions $\mathsf{element}_{\mathrm{I}} : \mathsf{set}_{\mathrm{I}} \rightarrow \mathsf{nat}$ for the first argument of $\mathsf{ins}_{\mathrm{I}}$ and $\mathsf{subset}_{\mathrm{I}} : \mathsf{set}_{\mathrm{I}} \rightarrow \mathsf{set}_{\mathrm{I}}$ for the second argument of $\mathsf{ins}_{\mathrm{I}}$. For these destructor functions we obtain the following representation axioms:

$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}_{\mathrm{I}}$
$\quad \mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{x}, \mathsf{B}) \rightarrow \mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))$

$\mathsf{element}_{\mathrm{I}}(\mathsf{empty}_{\mathrm{I}}) \equiv 0\ (\equiv \triangledown_{\mathsf{nat}})$

$\mathsf{subset}_{\mathrm{I}}(\mathsf{empty}_{\mathrm{I}}) \equiv \mathsf{empty}_{\mathrm{I}}$

$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}_{\mathrm{I}}$
$\quad \mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{x}, \mathsf{B}) \rightarrow \gamma_{\mathsf{ins}_{\mathrm{I}}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \equiv \mathsf{true}$

Now, $\mathsf{subset}_{\mathrm{I}}$ is 1-bounded with difference predicate $\Delta^{\mathrm{I}1}_{\mathsf{subset}_{\mathrm{I}}} : \mathsf{set}_{\mathrm{I}} \rightarrow \mathsf{bool}$, defined by

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad \Delta^{\mathrm{I1}}_{\mathsf{subset}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\end{array}$$

Furthermore, the function $\mathsf{term\_size}_{\mathsf{set}_{\mathrm{I}}} : \mathsf{set}_{\mathrm{I}} \to \mathsf{nat}$ is synthesized by:

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \to \mathsf{term\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{term\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{term\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{subset}_{\mathrm{I}}(\mathsf{A})))\end{array}$$

In order to have easier proofs, we specify a function $\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}} : \mathsf{set}_{\mathrm{I}} \to \mathsf{nat}$, by

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \to \mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \land \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \land \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{subset}_{\mathrm{I}}(\mathsf{A})))\end{array}$$

The specification of $\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}$ is case-distinct, as proved by

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad \neg\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \land\\ (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \land \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad \neg\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \land\\ (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \land \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad \neg\left(\begin{array}{c}(\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \land \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \land\\ (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \land \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\end{array}\right)\end{array}$$

Furthermore, the recursion ordering of $\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}$ is well-founded. To prove that we use the Estimation Calculus. There are two recursive cases with a single recursive call of $\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}$ in each. For the first recursive case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \land \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Then, using the Estimation Calculus, we obtain:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{subset}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \lor \Delta^{\mathrm{I1}}_{\mathsf{subset}_{\mathrm{I}}}(\mathsf{A})\right\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to (\mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{subset}_{\mathrm{I}}}(\mathsf{A}))\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})).\end{array}$$

Similarly, for the second recursive case, we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Then, using the Estimation Calculus, we obtain:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{subset}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{subset}_{\mathrm{I}}}(\mathsf{A})\right\rangle}\ \text{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to (\mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{subset}_{\mathrm{I}}}(\mathsf{A}))\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad (\mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{element}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{subset}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{A} \equiv \mathsf{ins}_{\mathrm{I}}(\mathsf{element}_{\mathrm{I}}(\mathsf{A}), \mathsf{subset}_{\mathrm{I}}(\mathsf{A})).\end{array}$$

Now, we need to prove that the above axiomatization of $\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}$ computes the minimal size of a set, indeed. Therefore we need to show the following proof obligations

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}:\mathsf{set}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \to (\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\ \exists\, \mathsf{B}:\mathsf{set}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \wedge (\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \geq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}:\mathsf{set}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \to \mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{B})\end{array}$$

Next, we need to show that ins denotes a size increasing constructor function. To do that, we prove:

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{set}_{\mathrm{I}}\\ \quad (\mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{set}_{\mathrm{I}}}(\mathsf{ins}_{\mathrm{I}}(\mathsf{x}, \mathsf{A}))) \equiv \mathsf{true}.\end{array}$$

Finally, we need to define the strictness predicate $\Theta^2_{\mathsf{ins}_\mathrm{I}} : \mathsf{nat} \times \mathsf{set}_\mathrm{I} \to \mathsf{bool}$ and the minimal representation predicate $\Gamma_{\mathsf{ins}_\mathrm{I}} : \mathsf{nat} \times \mathsf{set}_\mathrm{I} \to \mathsf{bool}$. We suggest the following definitions:

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set}_\mathrm{I}\\ \quad \Theta^2_{\mathsf{ins}_\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{x} \notin_\mathrm{I} \mathsf{A}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set}_\mathrm{I}\\ \quad \Gamma_{\mathsf{ins}_\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{x} \notin_\mathrm{I} \mathsf{A}\end{array}$$

However, we have to prove that our suggestions really define the strictness and the minimal representation predicate. Hence, we need to show that

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set}_\mathrm{I}\\ \quad \Theta^2_{\mathsf{ins}_\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow (\mathsf{min\_size}_{\mathsf{set}_\mathrm{I}}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{set}_\mathrm{I}}(\mathsf{ins}_\mathrm{I}(\mathsf{x},\mathsf{A}))) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set}_\mathrm{I}\\ \quad \Gamma_{\mathsf{ins}_\mathrm{I}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{min\_size}_{\mathsf{set}_\mathrm{I}}(\mathsf{ins}_\mathrm{I}(\mathsf{x},\mathsf{A})) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{set}_\mathrm{I}}(\mathsf{A}))\end{array}$$

Having done so, we know for our original specification set that the constructor function ins is size increasing, and we can translate the strictness predicate as well as the minimal representation predicate into the original specification. Hence, we obtain:

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set}\\ \quad \Theta^2_{\mathsf{ins}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{x} \notin \mathsf{A}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set}\\ \quad \Gamma_{\mathsf{ins}}(\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{x} \notin \mathsf{A}\end{array}$$

The data type set possesses non-overlapping constructor functions, since

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set}\\ \quad \mathsf{empty} \not\equiv \mathsf{ins}(\mathsf{x},\mathsf{A})\end{array}$$

holds. Hence, we can use the simplified construction scheme for the destructor functions.

For the constructor function ins we introduce two destructor functions $\mathsf{element} : \mathsf{set} \to \mathsf{nat}$ for the first argument of ins and $\mathsf{subset} : \mathsf{set} \to \mathsf{set}$ for the second argument of ins. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}{:}\,\mathsf{set}\\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{x},\mathsf{B}) \to \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{B}))\end{array}$$

$$\mathsf{element}(\mathsf{empty}) \equiv 0 \ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\mathsf{subset}(\mathsf{empty}) \equiv \mathsf{empty}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}{:}\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{x},\mathsf{B}) \\ \qquad \to \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \end{array}$$

The reflexive destructor function of the constructor function ins, subset, is 1-bounded, and the difference predicate $\Delta^1_{\mathsf{subset}} : \mathsf{set} \to \mathsf{bool}$ is defined by

$$\begin{array}{l} \forall\, \mathsf{A}{:}\mathsf{set} \\ \quad \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true} \\ \qquad \leftrightarrow \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \end{array}$$

For the data type set we will give constructive function and predicate specifications for delete, union, inter, diff, min, max, card, sort, $<_{\mathsf{set}}$, $\leq_{\mathsf{set}}$, $>_{\mathsf{set}}$, and $\geq_{\mathsf{set}}$.

8.1 delete : nat × set → set

delete computes the delete operation on sets, thus it removes a specified object in a set, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \to \mathsf{delete}(\mathsf{x},\mathsf{A}) \equiv \mathsf{empty} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{element}(\mathsf{A})) \\ \qquad \to \mathsf{delete}(\mathsf{x},\mathsf{A}) \equiv \mathsf{subset}(\mathsf{A}) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{element}(\mathsf{A})) \\ \qquad \to \mathsf{delete}(\mathsf{x},\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{delete}(\mathsf{x},\mathsf{subset}(\mathsf{A}))) \end{array}$$

The recursion ordering of delete is well-founded. There is only one definition case with a single recursive call of delete. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{element}(\mathsf{A}))$$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{element}(\mathsf{A})) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}\colon\mathsf{nat}\ \forall\, \mathsf{A}\colon\mathsf{set}\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{element}(\mathsf{A}))\\ \qquad \to \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})).\end{array}$$

In addition, delete denotes a 2-bounded function symbol. To prove this property, first of all, we need to show that delete is completely specified, i.e.,

$$\begin{array}{l}\forall\, \mathsf{x}\colon\mathsf{nat}\ \forall\, \mathsf{A}\colon\mathsf{set}\\ \quad \mathsf{A} \equiv \mathsf{empty} \vee\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{element}(\mathsf{A})) \vee\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{element}(\mathsf{A}))\end{array}$$

Then, we examine each definition case separately. For the first case we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \mathsf{A} \equiv \mathsf{empty}, \mathsf{empty} \preceq_{\mathsf{set}} \mathsf{empty}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \mathsf{A} \equiv \mathsf{empty}, \mathsf{empty} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the second case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{element}(\mathsf{A}))$$

by φ, and we obtain the derivation in the Estimation Calculus:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

And, for the third case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{element}(\mathsf{A}))$$

by φ. Furthermore, since this case is recursive, we can assume an additional inference rule as the induction hypothesis:

$$\xi \Rightarrow \dfrac{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \Delta\rangle}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{subset}(\mathsf{A})) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where ξ is an abbreviation for the formula

$$\forall\, \mathsf{x}\colon\mathsf{nat}\ \forall\, \mathsf{A}\colon\mathsf{set}\ \varphi \to \Delta$$

Then, we obtain the derivation:

$$\begin{array}{c}
- \\
\hline \text{Identity} \\
\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \rangle \\
\hline \text{Estimation} \\
\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true} \rangle \\
\hline \text{Induction Hypothesis} \\
\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{subset}(\mathsf{A})) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \rangle \\
\hline \text{Weak Embedding} \\
\left\langle \begin{array}{c} \varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{subset}(\mathsf{A}))) \preceq_{\mathsf{set}} \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})), \\ \Delta^2_{\mathsf{delete}}(\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle \\
\hline \text{Equation 3} \\
\left\langle \begin{array}{c} \varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{subset}(\mathsf{A}))) \preceq_{\mathsf{set}} \mathsf{A}, \\ \Delta^2_{\mathsf{delete}}(\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \vee \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle
\end{array}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set}\ \varphi \to \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{true}$$

needs to be proved.

In order to synthesize the difference predicate $\Delta^2_{\mathsf{delete}} : \mathsf{nat} \times \mathsf{set} \to \mathsf{bool}$, we use the simplified difference formulas from each derivation, and we obtain:

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{element}(\mathsf{A})) \\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{element}(\mathsf{A})) \\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{subset}(\mathsf{A})) \end{array}$$

8.2 union : set × set → set

union computes the union of two sets, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \to \mathsf{union}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{B} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \\ \qquad \to \mathsf{union}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{union}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \end{array}$$

The recursion ordering of union is well-founded. There is only one definition case with a single recursive call of union. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A}))$$

by φ, we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \\ \qquad \to \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})). \end{array}$$

8.3 inter : set × set → set

inter computes the intersection of two sets, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{empty} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B}) \\ \qquad \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B}) \\ \qquad \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \end{array}$$

The recursion ordering of inter is well-founded. There are two recursive definition cases with a single recursive call in each. For the first recursive case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B})$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set}\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B})\\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set}\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B})\\ \qquad \rightarrow \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})).\end{array}$$

For the second recursive case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B})$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set}\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B})\\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set}\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B})\\ \qquad \rightarrow \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})).\end{array}$$

In addition, inter denotes a 1-bounded function symbol. To prove this property, first of all, we need to show that inter is completely specified, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{set}\\ \quad \mathsf{A} \equiv \mathsf{empty} \vee\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B}) \vee\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B})\end{array}$$

Then, we examine each definition case separately. For the first case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{empty}$$

by φ, and we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set}} \mathsf{empty}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the second case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \land \mathsf{element}(\mathsf{A}) \in \mathsf{B})$$

by φ, and since this is a recursive case, we may assume

$$\xi \Rightarrow \quad \cfrac{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \Delta\rangle}{\langle \varphi, \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

as an additional inference rule, where ξ is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\ \varphi \to \Delta$$

Thus, we obtain the derivation

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}}{\left\langle \begin{array}{c} \varphi, \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \\ \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Induction Hypothesis}}{\left\langle \begin{array}{c} \varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \preceq_{\mathsf{set}} \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})), \\ \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \lor \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \preceq_{\mathsf{set}} \mathsf{A}, \\ \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \lor \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\ \varphi \to (\mathsf{false} \lor \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set}\ \varphi \to \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{true}$$

needs to be shown. For the third case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B})$$

by φ, and since this is a recursive case, we may assume

$$\xi \Rightarrow \quad \frac{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \Delta\rangle}{\langle \varphi, \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

as an additional inference rule, where ξ is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set}\ \varphi \to \Delta$$

Thus, we obtain the derivation

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}}{\left\langle \begin{array}{c} \varphi, \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \\ \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Induction Hypothesis}}{\left\langle \begin{array}{c} \varphi, \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})), \\ \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \Theta^2_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Strong Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{A}, \\ \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \Theta^2_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Equation 4}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved.

In order to synthesize the difference predicate $\Delta^1_{\mathsf{inter}} : \mathsf{nat} \times \mathsf{set} \to \mathsf{bool}$, we use the simplified difference formulas from each derivation, and we obtain:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \to \Delta^1_{\mathsf{inter}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B}) \\ \quad\quad \to \left(\begin{array}{l} \Delta^1_{\mathsf{inter}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \\ \quad\quad \leftrightarrow \left(\begin{array}{c} \Delta^1_{\mathsf{inter}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{inter}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \equiv \mathsf{false} \end{array} \right) \end{array} \right) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B}) \\ \qquad \to \Delta^1_{\mathsf{inter}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \end{array}$$

8.4 $\mathsf{diff} : \mathsf{set} \times \mathsf{set} \to \mathsf{set}$

diff computes the difference of two sets, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \to \mathsf{diff}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{empty} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B}) \\ \qquad \to \mathsf{diff}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B}) \\ \qquad \to \mathsf{diff}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \end{array}$$

The recursion ordering of diff is well-founded. There are two recursive definition cases with a single recursive call in each. For the first recursive case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B})$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \rangle}\,\mathsf{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\,\mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B}) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B}) \\ \qquad \to \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})). \end{array}$$

For the second recursive case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B})$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\,\text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\,\text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B}) \\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B}) \\ \qquad \rightarrow \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})). \end{array}$$

In addition, diff denotes a 1-bounded function symbol. To prove this property, first of all, we need to show that diff is completely specified, i.e.,

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}:\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \vee \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B}) \vee \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B}) \end{array}$$

Then, we examine each definition case separately. For the first case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{empty}$$

by φ, and we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set}} \mathsf{empty}, \mathsf{false}\rangle}\,\text{Identity}}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\,\text{Equation 1}$$

For the second case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B})$$

by φ, and since this is a recursive case, we may assume

$$\xi \Rightarrow \dfrac{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \Delta\rangle}{\langle \varphi, \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\,\text{Induction Hypothesis}$$

as an additional inference rule, where ξ is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\ \varphi \to \Delta$$

Thus, we obtain the derivation

$$\begin{array}{c}
- \\
\hline \text{Identity} \\ \hline
\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \rangle \\
\hline \text{Estimation} \\ \hline
\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true} \rangle \\
\hline \text{Induction Hypothesis} \\ \hline
\left\langle \begin{array}{c} \varphi, \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \\ \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \end{array} \right\rangle \\
\hline \text{Strong Embedding} \\ \hline
\left\langle \begin{array}{c} \varphi, \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})), \\ \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \Theta^2_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle \\
\hline \text{Equation 4} \\ \hline
\left\langle \begin{array}{c} \varphi, \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{A}, \\ \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \Theta^2_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle
\end{array}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})$$

has to e proved.

For the third case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \not\in \mathsf{B})$$

by φ, and since this is a recursive case, we may assume

$$\xi \Rightarrow \begin{array}{c}
\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \Delta \rangle \\
\hline \text{Induction Hypothesis} \\ \hline
\langle \varphi, \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \rangle
\end{array}$$

as an additional inference rule, where ξ is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\ \varphi \to \Delta$$

Thus, we obtain the derivation

$$\begin{array}{c}
- \\
\hline \text{Identity} \\
\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \rangle \\
\hline \text{Estimation} \\
\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true} \rangle \\
\hline \text{Induction Hypothesis} \\
\left\langle \begin{array}{c} \varphi, \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \\ \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \end{array} \right\rangle \\
\hline \text{Weak Embedding} \\
\left\langle \begin{array}{c} \varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \preceq_{\mathsf{set}} \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})), \\ \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \equiv \mathsf{false} \end{array} \right\rangle \\
\hline \text{Equation 3} \\
\left\langle \begin{array}{c} \varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \preceq_{\mathsf{set}} \mathsf{A}, \\ \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \equiv \mathsf{false} \end{array} \right\rangle
\end{array}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\ \varphi \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and where to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\ \varphi \rightarrow \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{true}$$

needs to be shown.

In order to synthesize the difference predicate $\Delta^1_{\mathsf{diff}} : \mathsf{nat} \times \mathsf{set} \rightarrow \mathsf{bool}$, we use the simplified difference formulas from each derivation, and we obtain:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \rightarrow \Delta^1_{\mathsf{diff}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \in \mathsf{B}) \\ \qquad \rightarrow \Delta^1_{\mathsf{diff}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \mathsf{element}(\mathsf{A}) \notin \mathsf{B}) \\ \qquad \rightarrow \left(\begin{array}{l} \Delta^1_{\mathsf{diff}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \\ \qquad \leftrightarrow \left(\begin{array}{c} \Delta^1_{\mathsf{diff}}(\mathsf{subset}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{diff}(\mathsf{subset}(\mathsf{A}), \mathsf{B})) \equiv \mathsf{false} \end{array} \right) \end{array} \right) \end{array}$$

8.5 $\mathsf{min} : \mathsf{set} \rightarrow \mathsf{nat}$

min computes the minimal element in a non-empty set, and it is defined by:

$$\forall\, \mathsf{A{:}set}\\ (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \wedge \mathsf{subset}(\mathsf{A}) \equiv \mathsf{nil})\\ \rightarrow \mathsf{min}(\mathsf{A}) \equiv \mathsf{element}(\mathsf{A})$$

$$\forall\, \mathsf{A{:}set}\\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge \\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix}\\ \rightarrow \mathsf{min}(\mathsf{A}) \equiv \mathsf{min}(\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{subset}(\mathsf{A}))))$$

$$\forall\, \mathsf{A{:}set}\\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge \\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix}\\ \rightarrow \mathsf{min}(\mathsf{A}) \equiv \mathsf{min}(\mathsf{subset}(\mathsf{A}))$$

The recursion ordering of min is well-founded. There are two definition cases with one recursive call in each. For the first recursive case we obtain the derivation in the Estimation Calculus, abbreviating the case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge \\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix}$$

by φ:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \begin{matrix} \varphi, \mathsf{subset}(\mathsf{subset}(\mathsf{A})) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{A}), \\ \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \end{matrix} \right\rangle}\ \text{Estimation}}{\left\langle \begin{matrix} \varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \preceq_{\mathsf{set}} \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})), \\ \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false} \end{matrix} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{matrix} \varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \preceq_{\mathsf{set}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false} \end{matrix} \right\rangle}\ \text{Equation 3}$$

where to apply the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A{:}set}\ \varphi \rightarrow \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true}$$

has to be shown. In order to ensure the strict relation, we, therefore, need to prove

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{set} \\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \land \\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})), \mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \land \\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right) \\ \quad \rightarrow \left(\begin{array}{c}\mathsf{false} \lor \Delta^1_{\mathsf{subset}}(\mathsf{subset}(\mathsf{A})) \equiv \mathsf{true} \lor \\ \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{set} \\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \land \\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})), \mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \land \\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right) \\ \quad \rightarrow \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})), \mathsf{subset}(\mathsf{subset}(\mathsf{A}))).\end{array}$$

For the second recursive case we abbreviate the case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \land \\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})), \mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \land \\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)$$

by φ, and we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{set} \\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \land \\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})), \mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \land \\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right) \\ \quad \rightarrow (\mathsf{false} \lor \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{set} \\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \land \\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})), \mathsf{subset}(\mathsf{subset}(\mathsf{A}))) \land \\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right) \\ \quad \rightarrow \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})).\end{array}$$

8.6 max : set → nat

max computes the maximal element in a non-empty set, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}\\ \quad (\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})) \wedge \mathsf{subset}(\mathsf{A}) \equiv \mathsf{nil})\\ \qquad \to \mathsf{max}(\mathsf{A}) \equiv \mathsf{element}(\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}\\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge\\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{true}\end{pmatrix}\\ \qquad \to \mathsf{max}(\mathsf{A}) \equiv \mathsf{max}(\mathsf{subset}(\mathsf{A}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}\\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge\\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false}\end{pmatrix}\\ \qquad \to \mathsf{max}(\mathsf{A}) \equiv \mathsf{max}(\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{subset}(\mathsf{A}))))\end{array}$$

The recursion ordering of max is well-founded. There are two definition cases with one recursive call in each. For the first recursive case we obtain the derivation in the Estimation Calculus, abbreviating the case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge\\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{true}\end{pmatrix}$$

by φ:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}\\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge\\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false}\end{pmatrix}\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}\\ \quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge\\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false}\end{pmatrix}\\ \qquad \to \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})).\end{array}$$

For the second recursive case we abbreviate the case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ \mathsf{subset}(\mathsf{A}) \equiv \mathsf{ins}(\mathsf{element}(\mathsf{subset}(\mathsf{A})),\mathsf{subset}(\mathsf{subset}(\mathsf{A})))\wedge\\ (\mathsf{element}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{element}(\mathsf{subset}(\mathsf{A}))) \equiv \mathsf{false}\end{pmatrix}$$

by φ, and we obtain the derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{subset(A)} \preceq_{\mathsf{set}} \mathsf{subset(A)}, \mathsf{false}\rangle}\;\mathsf{Identity}}{\left\langle \begin{array}{c}\varphi, \mathsf{subset(subset(A))} \preceq_{\mathsf{set}} \mathsf{subset(A)},\\ \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{subset(A)}) \equiv \mathsf{true}\end{array}\right\rangle}\;\mathsf{Estimation}}{\left\langle \begin{array}{c}\varphi, \mathsf{ins(element(A), subset(subset(A)))} \preceq_{\mathsf{set}} \mathsf{ins(element(A), subset(A))},\\ \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{subset(A)}) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{ins}}(\mathsf{element(A), subset(subset(A))}) \equiv \mathsf{false}\end{array}\right\rangle}\;\mathsf{Weak\ Embedding}}{\left\langle \begin{array}{c}\varphi, \mathsf{ins(element(A), subset(subset(A)))} \preceq_{\mathsf{set}} \mathsf{A},\\ \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{subset(A)}) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{ins}}(\mathsf{element(A), subset(subset(A))}) \equiv \mathsf{false}\end{array}\right\rangle}\;\mathsf{Equation\ 3}$$

where to enable the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}{:}\mathsf{set}\ \varphi \to \Gamma_{\mathsf{ins}}(\mathsf{element(A), subset(A)}) \equiv \mathsf{true}$$

has to be shown. In order to ensure the strict relation, we, therefore, need to prove

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{set}\\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{ins(element(A), subset(A))} \wedge\\ \mathsf{subset(A)} \equiv \mathsf{ins(element(subset(A)), subset(subset(A)))} \wedge\\ (\mathsf{element(A)} <_{\mathsf{nat}} \mathsf{element(subset(A))}) \equiv \mathsf{true}\end{array}\right)\\ \quad\to \left(\begin{array}{c}\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{subset(A)}) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{ins}}(\mathsf{element(A), subset(subset(A))}) \equiv \mathsf{false}\end{array}\right)\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{set}\\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{ins(element(A), subset(A))} \wedge\\ \mathsf{subset(A)} \equiv \mathsf{ins(element(subset(A)), subset(subset(A)))} \wedge\\ (\mathsf{element(A)} <_{\mathsf{nat}} \mathsf{element(subset(A))}) \equiv \mathsf{true}\end{array}\right)\\ \quad\to \mathsf{subset(A)} \equiv \mathsf{ins(element(subset(A)), subset(subset(A)))}.\end{array}$$

8.7 card : set → nat

card computes the cardinality of a set, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{set}\\ \quad \mathsf{A} \equiv \mathsf{empty} \to \mathsf{card(A)} \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{set}\\ \quad \mathsf{A} \equiv \mathsf{ins(element(A), subset(A))} \to \mathsf{card(A)} \equiv \mathsf{succ(card(subset(A)))}\end{array}$$

The recursion ordering of card is well-founded. There is only a single recursive definition case with a single recursive call. Hence, using the Estimation Calculus, abbreviating the invariant case condition

$$\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A}))$$

by φ, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \rightarrow \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \end{array}$$

8.8 sort : set → list

sort sorts a set, defined by:

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{empty} \rightarrow \mathsf{sort}(\mathsf{A}) \equiv \mathsf{nil} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{set} \\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \\ \qquad \rightarrow \mathsf{sort}(\mathsf{A}) \equiv \mathsf{cons}(\mathsf{min}(\mathsf{A}), \mathsf{sort}(\mathsf{delete}(\mathsf{min}(\mathsf{A}), \mathsf{A}))) \end{array}$$

The recursion ordering of sort is well-founded. There is only one recursive definition case with a single recursive call. Hence, we abbreviate the case condition

$$\mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A}))$$

by φ. Using the Estimation Calculus, we obtain the following derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{delete}(\mathsf{min}(\mathsf{A}), \mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{min}(\mathsf{A}), \mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{set}\\ \quad \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A}))\\ \qquad \rightarrow (\mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{min}(\mathsf{A}), \mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be proved by induction.

8.9 $<_{\mathsf{set}}$: set × set → bool

$<_{\mathsf{set}}$ computes the less-than-relation on lists, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad \mathsf{B} \equiv \mathsf{empty} \rightarrow (\mathsf{A} <_{\mathsf{set}} \mathsf{B}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad (\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{empty})\\ \qquad \rightarrow (\mathsf{A} <_{\mathsf{set}} \mathsf{B}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right)\\ \qquad \rightarrow (\mathsf{A} <_{\mathsf{set}} \mathsf{B}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right)\\ \qquad \rightarrow (\mathsf{A} <_{\mathsf{set}} \mathsf{B}) \equiv \mathsf{false}\end{array}$$

The recursion ordering of $<_{\mathsf{set}}$ is well-founded: There are two definition cases with a single recursive call of $<_{\mathsf{set}}$ in each. For each recursive definition case and each argument we use the Estimation Calculus. Starting with the first recursive case, we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right)$$

by φ. For the first argument of $<_{\mathsf{set}}$, A, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{set}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true}\end{array}\right)\\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{A}) \equiv \mathsf{true}),\end{array}$$

which can be simplified using the definition of $\Delta^1_{\mathsf{subset}}$ to

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad\left(\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ (\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{true}\end{array}\right)\\ \quad\to\mathsf{A}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A})).\end{array}$$

And for the second argument of $<_{\mathsf{set}}$, B, we obtain:

$$\dfrac{\dfrac{-}{\langle\varphi,\mathsf{B}\preceq_{\mathsf{set}}\mathsf{B},\mathsf{false}\rangle}\ \mathsf{Identity}}{\langle\varphi,\mathsf{subset}(\mathsf{B})\preceq_{\mathsf{set}}\mathsf{B},\mathsf{false}\vee\Delta^1_{\mathsf{subset}}(\mathsf{B})\equiv\mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{set}}$-relation, we have to show

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad\left(\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ (\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{true}\end{array}\right)\\ \quad\to(\mathsf{false}\vee\Delta^1_{\mathsf{subset}}(\mathsf{B})\equiv\mathsf{true}),\end{array}$$

which can be simplified using the definition of $\Delta^1_{\mathsf{subset}}$ to

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad\left(\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ (\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{true}\end{array}\right)\\ \quad\to\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B})).\end{array}$$

For the second recursive definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ (\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{false}\end{array}\right)$$

by φ. For the first argument of $<_{\mathsf{set}}$, A, we obtain:

$$\dfrac{\dfrac{-}{\langle\varphi,\mathsf{A}\preceq_{\mathsf{set}}\mathsf{A},\mathsf{false}\rangle}\ \mathsf{Identity}}{\langle\varphi,\mathsf{subset}(\mathsf{A})\preceq_{\mathsf{set}}\mathsf{A},\mathsf{false}\vee\Delta^1_{\mathsf{subset}}(\mathsf{A})\equiv\mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{set}}$-relation, we have to show

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad\left(\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ (\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{false}\end{array}\right)\\ \quad\to(\mathsf{false}\vee\Delta^1_{\mathsf{subset}}(\mathsf{A})\equiv\mathsf{true}),\end{array}$$

which can be simplified using the definition of $\Delta^1_{\mathsf{subset}}$ to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{set} \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \end{array}\right) \\ \quad \rightarrow \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})). \end{array}$$

And for the second argument of $<_{\mathsf{set}}$, B, we obtain:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{B} \preceq_{\mathsf{set}} \mathsf{B}, \mathsf{false} \rangle}\ \text{Identity}}{\langle \varphi, \mathsf{subset}(\mathsf{B}) \preceq_{\mathsf{set}} \mathsf{B}, \mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{B}) \equiv \mathsf{true} \rangle}\ \text{Estimation}$$

In order to ensure the strict $\prec_{\mathsf{set}}$-relation, we have to show

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{set} \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \end{array}\right) \\ \quad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{subset}}(\mathsf{B}) \equiv \mathsf{true}), \end{array}$$

which can be simplified using the definition of $\Delta^1_{\mathsf{subset}}$ to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{set} \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \end{array}\right) \\ \quad \rightarrow \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})). \end{array}$$

Thus, the recursion ordering of $<_{\mathsf{set}}$ is a well-founded ordering.

In addition, $<_{\mathsf{set}}$ denotes a well-founded ordering as well. To prove that, we first have to show that $<_{\mathsf{set}}$ is completely specified, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{set} \\ \quad (\mathsf{B} \equiv \mathsf{empty}) \vee \\ \quad (\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{empty}) \vee \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true} \end{array}\right) \vee \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \end{array}\right) \end{array}$$

Next, for each definition case we show that

$$\forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{set}\ (\mathsf{A} <_{\mathsf{set}} \mathsf{B}) \equiv \mathsf{true} \rightarrow \mathsf{A} \prec_{\mathsf{set}} \mathsf{B},$$

again, using the Estimation Calculus. For the first case we obtain

$$\cfrac{-}{\langle \mathsf{B} \equiv \mathsf{empty} \wedge \mathsf{false} \equiv \mathsf{true}, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{B}, \Delta_1 \rangle}\ \text{Tautology}$$

where in order to enable the application of the Tautology Rule, the first-order formula

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad\neg(\mathsf{B}\equiv\mathsf{empty}\wedge\mathsf{false}\equiv\mathsf{true})\end{array}$$

has to be proved. To prove the strict relation, the formula

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad(\mathsf{B}\equiv\mathsf{empty}\wedge\mathsf{false}\equiv\mathsf{true})\rightarrow\Delta_1\end{array}$$

has to be shown. For the second case we obtain the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\left\langle\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{empty}\wedge\mathsf{true}\equiv\mathsf{true},\\ \mathsf{empty}\preceq_{\mathsf{set}}\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B})),\mathsf{true}\end{array}\right\rangle}\ \text{Strong Estimation}}{\left\langle\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{empty}\wedge\mathsf{true}\equiv\mathsf{true},\\ \mathsf{empty}\preceq_{\mathsf{set}}\mathsf{B},\mathsf{true}\end{array}\right\rangle}\ \text{Equation 1}}{\left\langle\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{empty}\wedge\mathsf{true}\equiv\mathsf{true},\\ \mathsf{A}\preceq_{\mathsf{set}}\mathsf{B},\mathsf{true}\end{array}\right\rangle}\ \text{Equation 5}}{}$$

showing the strict relation by

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{set}\\ \quad(\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{empty}\wedge\mathsf{true}\equiv\mathsf{true})\\ \qquad\rightarrow\mathsf{true}\end{array}$$

The third definition case is a recursive case. Hence, we need to make an additional case analysis:

$(\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{true}$ or

$(\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{false}.$

For the first case we can assume as an induction hypothesis the inference rule:

$$\dfrac{-}{\langle\varphi,\mathsf{subset}(\mathsf{A})\preceq_{\mathsf{set}}\mathsf{subset}(\mathsf{B}),\mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where we use φ as an abbreviation for

$$\left(\begin{array}{c}\mathsf{B}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B}))\wedge\mathsf{A}\equiv\mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge\\ (\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{true}\wedge\mathsf{true}\equiv\mathsf{true}\wedge\\ (\mathsf{subset}(\mathsf{A})<_{\mathsf{set}}\mathsf{subset}(\mathsf{B}))\equiv\mathsf{true}\end{array}\right)$$

Then, the derivation of

$\langle\varphi,\mathsf{A}\preceq_{\mathsf{set}}\mathsf{B},\Delta_3\rangle$

is achieved by:

$$
\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{subset}(\mathsf{A}) \preceq_{\mathsf{set}} \mathsf{subset}(\mathsf{B}), \mathsf{true}\rangle}\ \text{Induction Hypothesis}}{\left\langle \begin{array}{c}\varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \preceq_{\mathsf{set}} \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})),\\ \mathsf{true} \vee \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{false}\end{array}\right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c}\varphi, \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \preceq_{\mathsf{set}} \mathsf{B},\\ \mathsf{true} \vee \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{false}\end{array}\right\rangle}\ \text{Equation 3}}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{B}, \mathsf{true} \vee \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{false}\rangle}\ \text{Equation 5}
$$

where in order to enable the application of the Weak Embedding Rule, the first-order formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\ \varphi \to \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true}$$

has to be shown. The strict relation is proved by

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\\ \quad \varphi \to (\mathsf{true} \vee \Gamma_{\mathsf{ins}}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \equiv \mathsf{false}).\end{array}$$

For the second case we cannot assume an induction hypothesis. We prove the estimation formula

$$\left\langle \begin{array}{c}\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false}, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{B}, \Delta_4\end{array}\right\rangle$$

by an application of the Tautology Rule, where in order to enable this application, it is necessary to prove the first-order formula:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\\ \quad \neg\left(\begin{array}{c}\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right)\end{array}$$

To prove the strict relation, the formula

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set}\\ \quad \left(\begin{array}{c}\mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}), \mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}), \mathsf{subset}(\mathsf{A})) \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true} \wedge \mathsf{true} \equiv \mathsf{true} \wedge\\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false}\end{array}\right)\\ \quad \to \Delta_4\end{array}$$

needs to be shown.

The fourth definition case is also a recursive case. Hence, we need to make an additional case analysis:

$(\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true}$ or

$(\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false}.$

Although for the first case we could assume an induction hypothesis, this is not necessary since the derivation of the estimation formula

$$\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true}\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true}, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{B}, \Delta_5 \end{array} \right\rangle$$

can be achieved by the application of the Tautology Rule. In order to enable this application, the first-order formula

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad \neg \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true}\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true} \end{array} \right) \end{array}$$

has to be shown. And for the strict relation, the formula

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true}\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{true} \end{array} \right) \\ \quad \to \Delta_5 \end{array}$$

needs to be proved. For the second case we cannot assume an induction hypothesis. We prove the estimation formula

$$\left\langle \begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true}\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false}, \mathsf{A} \preceq_{\mathsf{set}} \mathsf{B}, \Delta_6 \end{array} \right\rangle$$

by an application of the Tautology Rule, where in order to enable this application, it is necessary to prove the first-order formula:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad \neg \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true}\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \end{array} \right) \end{array}$$

To prove the strict relation, the formula

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set} \\ \quad \left(\begin{array}{c} \mathsf{B} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{B}),\mathsf{subset}(\mathsf{B})) \wedge \mathsf{A} \equiv \mathsf{ins}(\mathsf{element}(\mathsf{A}),\mathsf{subset}(\mathsf{A}))\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \wedge \mathsf{false} \equiv \mathsf{true}\wedge \\ (\mathsf{subset}(\mathsf{A}) <_{\mathsf{set}} \mathsf{subset}(\mathsf{B})) \equiv \mathsf{false} \end{array} \right) \\ \quad \to \Delta_6 \end{array}$$

needs to be shown.

Having proved all these obligations, $<_{\mathsf{set}}$ denotes a well-founded order relation.

8.10 $\leq_{\mathsf{set}}$: set × set → bool

$\leq_{\mathsf{set}}$ computes the less-than-or-equal-relation on sets, and it is defined by:

$$\begin{array}{l} \forall\ \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad (\mathsf{A} \leq_{\mathsf{set}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} <_{\mathsf{set}} \mathsf{A}) \equiv \mathsf{false} \end{array}$$

Since this is a non-recursive constructive definition we are done.

8.11 $>_{\mathsf{set}}$: set × set → bool

$>_{\mathsf{set}}$ computes the greater-than-relation on sets, and it is defined by:

$$\begin{array}{l} \forall\ \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad (\mathsf{A} >_{\mathsf{set}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} <_{\mathsf{set}} \mathsf{A}) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive constructive definition we are done.

8.12 $\geq_{\mathsf{set}}$: set × set → bool

$\geq_{\mathsf{set}}$ computes the greater-than-relation on sets, and it is defined by:

$$\begin{array}{l} \forall\ \mathsf{A}, \mathsf{B} : \mathsf{set} \\ \quad (\mathsf{A} \geq_{\mathsf{set}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} \leq_{\mathsf{set}} \mathsf{A}) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive constructive definition we are done.

9

Finite Sets, set2

This specification of finite sets (of nats), set2, uses three constructor functions $\mathsf{empty} :\rightarrow \mathsf{set2}$, generating the empty set, $\mathsf{single} : \mathsf{nat} \rightarrow \mathsf{set2}$, generating a singleton set, and $\mathsf{union} : \mathsf{set2} \times \mathsf{set2} \rightarrow \mathsf{set2}$, for the union of two sets. Equality on set2 is specified using an auxiliary predicate $\in : \mathsf{nat} \times \mathsf{set2} \rightarrow \mathsf{bool}$ by the axioms:

$$\forall\, x : \mathsf{nat}\ \ x \notin \mathsf{empty}$$

$$\forall\, x, y : \mathsf{nat}\ \ x \in \mathsf{single}(y) \leftrightarrow x \equiv y$$

$$\forall\, x : \mathsf{nat}\ \forall\, A, B : \mathsf{set2}\ (x \in \mathsf{union}(A, B)) \leftrightarrow (x \in A \vee x \in B)$$

$$\forall\, A, B : \mathsf{set2} \quad A \equiv B \leftrightarrow (\forall\, x : \mathsf{nat}\ x \in A \leftrightarrow x \in B)$$

By the above specification we have defined a non-freely generated data type. Hence, we must prove the constructor function union to be size increasing by using the respective implementation specification. Furthermore, the strictness predicates $\Theta^1_{\mathsf{union}} : \mathsf{set2} \times \mathsf{set2} \rightarrow \mathsf{bool}$ and $\Theta^2_{\mathsf{union}} : \mathsf{set2} \times \mathsf{set2} \rightarrow \mathsf{bool}$, as well as the minimal representation predicate $\Gamma_{\mathsf{union}} : \mathsf{set2} \times \mathsf{set2} \rightarrow \mathsf{bool}$ have to be synthesized.

The implementation specification is automatically generated using the constructor functions $\mathsf{empty}_{\mathrm{I}} :\rightarrow \mathsf{set2}_{\mathrm{I}}$, $\mathsf{single}_{\mathrm{I}} : \mathsf{nat} \rightarrow \mathsf{set2}_{\mathrm{I}}$, $\mathsf{union}_{\mathrm{I}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \rightarrow \mathsf{set2}_{\mathrm{I}}$, and the new equality predicate $\mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \rightarrow \mathsf{bool}$.

$$\forall\, x : \mathsf{nat}$$
$$\mathsf{empty}_{\mathrm{I}} \not\equiv \mathsf{single}_{\mathrm{I}}(x)$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \mathsf{empty}_{\mathrm{I}} \not\equiv \mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \mathsf{single}_{\mathrm{I}}(\mathsf{x}) \not\equiv \mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})$$

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat} \quad \mathsf{single}_{\mathrm{I}}(\mathsf{x}) \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{y}) \to \mathsf{x} \equiv \mathsf{y}$$

$$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D} : \mathsf{set2}_{\mathrm{I}} \quad \mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{C}, \mathsf{D}) \to (\mathsf{A} \equiv \mathsf{C} \wedge \mathsf{B} \equiv \mathsf{D})$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{x} \notin_{\mathrm{I}} \mathsf{empty}_{\mathrm{I}}$$

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \mathsf{x} \in \mathsf{single}_{\mathrm{I}}(\mathsf{y}) \leftrightarrow \mathsf{x} \equiv \mathsf{y}$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}}\ (\mathsf{x} \in_{\mathrm{I}} \mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})) \leftrightarrow (\mathsf{x} \in_{\mathrm{I}} \mathsf{A} \vee \mathsf{x} \in_{\mathrm{I}} \mathsf{B})$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{x} \in_{\mathrm{I}} \mathsf{A} \leftrightarrow \mathsf{x} \in_{\mathrm{I}} \mathsf{B})$$

$$\forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \quad \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{A}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \to \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{A}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set2}_{\mathrm{I}} \quad (\mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{B}, \mathsf{C}) \equiv \mathsf{true}) \to \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{C}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad (\mathsf{x} \equiv \mathsf{y} \wedge \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}) \to (\mathsf{x} \in_{\mathrm{I}} \mathsf{A} \leftrightarrow \mathsf{y} \in_{\mathrm{I}} \mathsf{B})$$

Since $\mathsf{set2}_{\mathrm{I}}$ is freely generated, the strictness predicates $\theta^1_{\mathsf{union}_{\mathrm{I}}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \to \mathsf{bool}$ and $\theta^2_{\mathsf{union}_{\mathrm{I}}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \to \mathsf{bool}$, as well as the minimal representation predicate $\gamma_{\mathsf{union}_{\mathrm{I}}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \to \mathsf{bool}$ are defined by:

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \theta^1_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \theta^2_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \gamma_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

In addition, all constructor functions of $\mathsf{set2}_{\mathrm{I}}$ are non-overlapping. Hence, the destructor function $\mathsf{get\_nat}_{\mathrm{I}} : \mathsf{set2}_{\mathrm{I}} \to \mathsf{nat}$ for the constructor function $\mathsf{single}_{\mathrm{I}}$ is defined by

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \quad \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{x}) \to \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})),$$

$\mathsf{get\_nat}_I(\mathsf{empty}_I) \equiv 0\ (\equiv \triangledown_{\mathsf{nat}})$

$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_I$
$\quad \mathsf{get\_nat}_I(\mathsf{union}_I(\mathsf{A}, \mathsf{B})) \equiv 0\ (\equiv \triangledown_{\mathsf{nat}})$

And for the constructor function union_I we introduce two destructor functions $\mathsf{left\_set}_I : \mathsf{set2}_I \to \mathsf{set2}_I$ for the first argument of union_I and $\mathsf{right\_set}_I : \mathsf{set2}_I \to \mathsf{set2}_I$ for the second argument of union_I. For these destructor functions we obtain the following representation axioms:

$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set2}_I$
$\quad \mathsf{A} \equiv \mathsf{union}_I(\mathsf{B}, \mathsf{C}) \to \mathsf{A} \equiv \mathsf{union}_I(\mathsf{left\_set}_I(\mathsf{A}), \mathsf{right\_set}_I(\mathsf{A}))$

$\mathsf{left\_set}_I(\mathsf{empty}_I) \equiv \mathsf{empty}_I$

$\mathsf{right\_set}_I(\mathsf{empty}_I) \equiv \mathsf{empty}_I$

$\forall\, \mathsf{x} : \mathsf{nat}$
$\quad \mathsf{left\_set}_I(\mathsf{single}_I(\mathsf{x})) \equiv \mathsf{single}_I(\mathsf{x})$

$\forall\, \mathsf{x} : \mathsf{nat}$
$\quad \mathsf{right\_set}_I(\mathsf{single}_I(\mathsf{x})) \equiv \mathsf{single}_I(\mathsf{x})$

$\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set2}_I$
$\quad \mathsf{A} \equiv \mathsf{union}_I(\mathsf{B}, \mathsf{C}) \to \gamma_{\mathsf{union}_I}(\mathsf{left\_set}_I(\mathsf{A}), \mathsf{right\_set}_I(\mathsf{A})) \equiv \mathsf{true}$

Now, $\mathsf{left\_set}_I$ and $\mathsf{right\_set}_I$ are both 1-bounded with difference predicates $\Delta^{I1}_{\mathsf{left\_set}_I} : \mathsf{set2}_I \to \mathsf{bool}$ and $\Delta^{I2}_{\mathsf{right\_set}_I} : \mathsf{set2}_I \to \mathsf{bool}$, defined by

$\forall\, \mathsf{A} : \mathsf{set2}_I$
$\quad \Delta^{I1}_{\mathsf{left\_set}_I}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{union}_I(\mathsf{left\_set}_I(\mathsf{A}), \mathsf{right\_set}_I(\mathsf{A}))$

$\forall\, \mathsf{A} : \mathsf{set2}_I$
$\quad \Delta^{I1}_{\mathsf{right\_set}_I}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{union}_I(\mathsf{left\_set}_I(\mathsf{A}), \mathsf{right\_set}_I(\mathsf{A}))$

Furthermore, the function $\mathsf{term\_size}_{\mathsf{set2}_I} : \mathsf{set2}_I \to \mathsf{nat}$ is synthesized by:

$\forall\, \mathsf{A} : \mathsf{set2}_I$
$\quad \mathsf{A} \equiv \mathsf{empty}_I \to \mathsf{term\_size}_{\mathsf{set2}_I}(\mathsf{A}) \equiv 0$

$\forall\, \mathsf{A} : \mathsf{set2}_I$
$\quad \mathsf{A} \equiv \mathsf{single}_I(\mathsf{get\_nat}_I(\mathsf{A})) \to \mathsf{term\_size}_{\mathsf{set2}_I}(\mathsf{A}) \equiv 0$

$\forall\, \mathsf{A} : \mathsf{set2}_I$
$\quad \mathsf{A} \equiv \mathsf{union}_I(\mathsf{left\_set}_I(\mathsf{A}), \mathsf{right\_set}_I(\mathsf{A}))$
$\qquad \to \mathsf{term\_size}_{\mathsf{set2}_I}(\mathsf{A}) \equiv$
$\qquad\qquad \mathsf{succ}(\mathsf{term\_size}_{\mathsf{set2}_I}(\mathsf{left\_set}_I(\mathsf{A})) + \mathsf{term\_size}_{\mathsf{set2}_I}(\mathsf{right\_set}_I(\mathsf{A})))$

In order to have easier proofs, we specify a function $\mathsf{min\_size}_{\mathsf{set2}_I} : \mathsf{set2}_I \to \mathsf{nat}$, by

$\forall\, \mathsf{A} : \mathsf{set2}_I$
$\quad \mathsf{min\_size}_{\mathsf{set2}_I}(\mathsf{A}) \equiv \mathsf{pred}(\mathsf{card}(\mathsf{A}))$,

where we use the following auxiliary functions, $\mathsf{card} : \mathsf{set2_I} \to \mathsf{nat}$ computing the cardinality of a set and $\mathsf{inter} : \mathsf{set2_I} \times \mathsf{set2_I} \to \mathsf{set2_I}$ computing the intersection of two sets, defined by

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2_I} \\ \quad \mathsf{A} \equiv \mathsf{empty_I} \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{empty_I}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2_I} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{single_I}(\mathsf{get\_nat_I}(\mathsf{A})) \wedge \\ \mathsf{get\_nat_I}(\mathsf{A}) \in_\mathrm{I} \mathsf{B} \end{array} \right) \\ \quad\quad \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{A}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2_I} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{single_I}(\mathsf{get\_nat_I}(\mathsf{A})) \wedge \\ \mathsf{get\_nat_I}(\mathsf{A}) \notin_\mathrm{I} \mathsf{B} \end{array} \right) \\ \quad\quad \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{empty_I}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2_I} \\ \quad \mathsf{A} \equiv \mathsf{union_I}(\mathsf{left\_set_I}(\mathsf{A}), \mathsf{right\_set_I}(\mathsf{A})) \\ \quad\quad \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{union_I}(\mathsf{inter}(\mathsf{left\_set_I}(\mathsf{A}), \mathsf{B}), \mathsf{inter}(\mathsf{right\_set_I}(\mathsf{A}), \mathsf{B}))\end{array}$$

and

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2_I} \\ \quad \mathsf{A} \equiv \mathsf{empty_I} \to \mathsf{card}(\mathsf{A}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2_I} \\ \quad \mathsf{A} \equiv \mathsf{single_I}(\mathsf{get\_nat_I}(\mathsf{A})) \to \mathsf{card}(\mathsf{A}) \equiv \mathsf{succ}(0)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2_I} \\ \quad \mathsf{A} \equiv \mathsf{union_I}(\mathsf{left\_set_I}(\mathsf{A}), \mathsf{right\_set_I}(\mathsf{A})) \\ \quad\quad \to \mathsf{card}(\mathsf{A}) \equiv \\ \quad\quad\quad \left(\begin{array}{l} (\mathsf{card}(\mathsf{left\_set_I}(\mathsf{A})) + \mathsf{card}(\mathsf{right\_set_I}(\mathsf{A}))) \\ \quad - \mathsf{card}(\mathsf{inter}(\mathsf{left\_set_I}(\mathsf{A}), \mathsf{right\_set_I}(\mathsf{A}))) \end{array} \right)\end{array}$$

Both specifications are case-distinct, as proved by

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2_I} \\ \quad \neg \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{empty_I} \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{single_I}(\mathsf{get\_nat_I}(\mathsf{A})) \wedge \\ \mathsf{get\_nat_I}(\mathsf{A}) \in_\mathrm{I} \mathsf{B} \end{array} \right) \end{array} \right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2_I} \\ \quad \neg \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{empty_I} \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{single_I}(\mathsf{get\_nat_I}(\mathsf{A})) \wedge \\ \mathsf{get\_nat_I}(\mathsf{A}) \notin_\mathrm{I} \mathsf{B} \end{array} \right) \end{array} \right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2_I} \\ \quad \neg \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{empty_I} \wedge \\ \mathsf{A} \equiv \mathsf{union_I}(\mathsf{left\_set_I}(\mathsf{A}), \mathsf{right\_set_I}(\mathsf{A})) \end{array} \right)\end{array}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \neg \left(\begin{pmatrix} \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{B} \end{pmatrix} \wedge \\ \begin{pmatrix} \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{B} \end{pmatrix} \right)$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \neg \left(\begin{pmatrix} \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{B} \end{pmatrix} \wedge \\ \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \right)$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \neg \left(\begin{pmatrix} \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{B} \end{pmatrix} \wedge \\ \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \right)$$

and

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \neg \begin{pmatrix} \mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \wedge \\ \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \end{pmatrix}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \neg \begin{pmatrix} \mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \wedge \\ \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \end{pmatrix}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \quad \neg \begin{pmatrix} \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \end{pmatrix}$$

Furthermore, both recursion orderings of inter and of card are well-founded. To prove that we use the Estimation Calculus. In the specification of inter there is only one recursive case with two recursive calls of inter. Now, we abbreviate the case condition

$$\mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Then, for the first recursive call the derivation in the Estimation Calculus is given by

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \rangle}\;\mathsf{Identity}}{\left\langle \varphi, \mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}) \right\rangle}\;\mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \rightarrow (\mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}))$$

which can be simplified to

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{set2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})).\end{array}$$

Similarly, for the second recursive call we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{right\_set}_{\mathrm{I}}}(\mathsf{A})\right\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{set2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to (\mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{right\_set}_{\mathrm{I}}}(\mathsf{A}))\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{set2}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))\\ \qquad \to \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})).\end{array}$$

In addition, inter is a 1-bounded function symbol. To prove that, first of all, we need to show that inter is completely specified, by proving:

$$\begin{array}{l}\forall\ \mathsf{A}, \mathsf{B}:\mathsf{set2}_{\mathrm{I}}\\ \qquad\qquad \mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \vee\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge\\ \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{B}\end{array}\right) \vee\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge\\ \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{B}\end{array}\right) \vee\\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))\end{array}$$

Now, we examine each definition case separately. For the first definition case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}}$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty}_{\mathrm{I}} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{empty}_{\mathrm{I}}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{empty}_{\mathrm{I}} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Equation\ 1}$$

For the second definition case we abbreviate the invariant case condition

$$\left(\begin{array}{l}\mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}))\wedge \\ \qquad \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{B}\end{array}\right)$$

by φ, and, using the Estimation Calculus, we obtain:

$$\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}$$

For the third definition case we abbreviate the invariant case condition

$$\left(\begin{array}{l}\mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}))\wedge \\ \qquad \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{B}\end{array}\right)$$

by φ, and, using the Estimation Calculus, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty}_{\mathrm{I}} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})), \mathsf{false}\rangle}\ \text{Equivalence}}{\langle \varphi, \mathsf{empty}_{\mathrm{I}} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the fourth definition case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Since this is a recursive case, we may assume the following induction hypotheses as additional inference rules:

$$\xi_1 \Rightarrow \quad \dfrac{\langle \varphi, \mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \Delta_1\rangle}{\langle \varphi, \mathsf{inter}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where ξ_1 is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}}\ \varphi \rightarrow \Delta_1$$

and

$$\xi_2 \Rightarrow \quad \dfrac{\langle \varphi, \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \Delta_2\rangle}{\langle \varphi, \mathsf{inter}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where ξ_2 is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}}\ \varphi \rightarrow \Delta_2$$

Thus, we obtain

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{False}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{False} \vee \Delta^1_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{inter}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}}\ \varphi \to (\mathsf{False} \vee \Delta^1_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved. On the other hand, we obtain:

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{False}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{False} \vee \Delta^1_{\mathsf{right\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{inter}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where to allow the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}}\ \varphi \to (\mathsf{False} \vee \Delta^1_{\mathsf{right\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true})$$

needs to be shown. Having derived the above two estimation formulas, we can now continue the derivation:

$$\dfrac{\dfrac{\begin{array}{c}\langle \varphi, \mathsf{inter}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle\,, \\ \langle \varphi, \mathsf{inter}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle\end{array}}{\left\langle \begin{array}{c} \varphi, \mathsf{union}_{\mathrm{I}}(\mathsf{inter}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}), \mathsf{inter}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B})) \\ \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})), \\ \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{union}_{\mathrm{I}}}(\mathsf{inter}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}), \mathsf{inter}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B})) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{union}_{\mathrm{I}}(\mathsf{inter}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}), \mathsf{inter}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B})) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \\ \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{union}_{\mathrm{I}}}(\mathsf{inter}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}), \mathsf{inter}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B})) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where in order to apply the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}}\ \varphi \to \Gamma_{\mathsf{union}_{\mathrm{I}}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \equiv \mathsf{true}$$

has to be shown.

Now, we have proved that inter is 1-bounded, and the respective difference predicate $\Delta^1_{\mathsf{inter}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \to \mathsf{bool}$ is synthesized using the simplified difference formulas from each call of the Estimation Calculus:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \to \Delta^1_{\mathsf{inter}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \qquad \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{B} \end{array} \right) \\ \qquad \to \Delta^1_{\mathsf{inter}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \qquad \mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{B} \end{array} \right) \\ \qquad \to \Delta^1_{\mathsf{inter}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \\ \qquad \to \left(\begin{array}{l} \Delta^1_{\mathsf{inter}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \\ \qquad \leftrightarrow \left(\begin{array}{l} \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \vee \\ \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true} \end{array} \right) \end{array} \right) \end{array}$$

which can be further simplified to

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \Delta^1_{\mathsf{inter}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \end{array}$$

Using that inter is 1-bounded we can prove that the recursion ordering of card is well-founded. There is only one recursive definition case with three recursive calls of card. Hence, we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Using the Estimation Calculus for the first recursive call we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true} \right\rangle}\ \mathsf{Estimation}$$

In order to show the strict relation, we need to prove

$$\begin{array}{l} \forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \\ \qquad \rightarrow \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))\end{array}$$

For the second recursive call of card we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

In order to show the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{right\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \\ \qquad \rightarrow \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))\end{array}$$

And for the third recursive call of card we obtain the derivation:

$$\cfrac{\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}}{\left\langle \begin{array}{c}\varphi, \mathsf{inter}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \preceq_{\mathsf{set2}_{\mathrm{I}}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true} \vee \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right\rangle}\ \mathsf{Estimation}$$

In order to show the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \\ \qquad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{left\_set}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true} \vee \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A})) \\ \qquad \rightarrow \mathsf{A} \equiv \mathsf{union}_{\mathrm{I}}(\mathsf{left\_set}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_set}_{\mathrm{I}}(\mathsf{A}))\end{array}$$

Now, we need to prove that the above axiomatization of $\mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}$ computes the minimal size of a set, indeed. Therefore we need to show the following proof obligations

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \rightarrow (\mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2}_{\mathrm{I}}\ \exists\, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \wedge (\mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}) \geq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \mathsf{Eq}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \rightarrow \mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{B})\end{array}$$

Next, we need to show that union denotes a size increasing constructor function. To do that, we prove:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \qquad (\mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}))) \equiv \mathsf{true} \wedge \\ \qquad (\mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{B}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}\end{array}$$

Finally, we need to define the strictness predicates $\Theta^1_{\mathsf{union}_{\mathrm{I}}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \rightarrow \mathsf{bool}$ and $\Theta^2_{\mathsf{union}_{\mathrm{I}}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \rightarrow \mathsf{bool}$, as well as the minimal representation predicate $\Gamma_{\mathsf{union}_{\mathrm{I}}} : \mathsf{set2}_{\mathrm{I}} \times \mathsf{set2}_{\mathrm{I}} \rightarrow \mathsf{bool}$. We suggest the following definitions:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \Theta^1_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \\ \qquad \leftrightarrow \left[\begin{array}{l}(\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{x} \in_{\mathrm{I}} \mathsf{B} \rightarrow \mathsf{x} \in_{\mathrm{I}} \mathsf{A}) \vee \\ (\exists\, \mathsf{y} : \mathsf{nat}\ \mathsf{A} \equiv \mathsf{empty}_{\mathrm{I}} \wedge \mathsf{B} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{y}))\end{array}\right]\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \Theta^2_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{false} \\ \qquad \leftrightarrow \left[\begin{array}{l}(\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{x} \in_{\mathrm{I}} \mathsf{A} \rightarrow \mathsf{x} \in_{\mathrm{I}} \mathsf{B}) \vee \\ (\exists\, \mathsf{y} : \mathsf{nat}\ \mathsf{B} \equiv \mathsf{empty}_{\mathrm{I}} \wedge \mathsf{A} \equiv \mathsf{single}_{\mathrm{I}}(\mathsf{y}))\end{array}\right]\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \Gamma_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \\ \qquad \leftrightarrow \left(\begin{array}{l}\mathsf{A} \not\equiv \mathsf{empty}_{\mathrm{I}} \wedge \mathsf{B} \not\equiv \mathsf{empty}_{\mathrm{I}} \wedge \\ (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{x} \notin_{\mathrm{I}} \mathsf{A} \vee \mathsf{x} \notin_{\mathrm{I}} \mathsf{B})\end{array}\right)\end{array}$$

However, we have to prove that our suggestions really define the strictness predicates and the minimal representation predicate. Hence, we need to show that

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \Theta^1_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \Theta^2_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{B}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}_{\mathrm{I}} \\ \quad \Gamma_{\mathsf{union}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \\ \qquad \leftrightarrow \mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{union}_{\mathrm{I}}(\mathsf{A}, \mathsf{B})) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{A}) + \mathsf{min\_size}_{\mathsf{set2}_{\mathrm{I}}}(\mathsf{B}))\end{array}$$

Having done so, we know for our original specification set2 that the constructor function union is size increasing, and we can translate the strictness predicates and the minimal representation predicate into the original specification. Hence, we obtain:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2}\\ \quad \Theta^1_{\mathsf{union}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{false}\\ \qquad \leftrightarrow \left[\begin{array}{l}(\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{x} \in \mathsf{B} \to \mathsf{x} \in \mathsf{A}) \vee\\ (\exists\, \mathsf{y}\!:\!\mathsf{nat}\ \mathsf{A} \equiv \mathsf{empty} \wedge \mathsf{B} \equiv \mathsf{single}(\mathsf{y}))\end{array}\right]\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2}\\ \quad \Theta^2_{\mathsf{union}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{false}\\ \qquad \leftrightarrow \left[\begin{array}{l}(\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{x} \in \mathsf{A} \to \mathsf{x} \in \mathsf{B}) \vee\\ (\exists\, \mathsf{y}\!:\!\mathsf{nat}\ \mathsf{B} \equiv \mathsf{empty} \wedge \mathsf{A} \equiv \mathsf{single}(\mathsf{y}))\end{array}\right]\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2}\\ \quad \Gamma_{\mathsf{union}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{l}\mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{B} \not\equiv \mathsf{empty} \wedge\\ (\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{x} \notin \mathsf{A} \vee \mathsf{x} \notin \mathsf{B})\end{array}\right)\end{array}$$

The data type set2 possesses overlapping constructor functions, since

$$\mathsf{empty} \equiv \mathsf{union}(\mathsf{empty}, \mathsf{empty})$$

Thus, we cannot use the simplified construction scheme for the destructor functions.

The destructor function get_nat : set2 $\to$ nat for the constructor function single is defined by:

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set2}\\ \quad \mathsf{A} \equiv \mathsf{single}(\mathsf{x}) \to \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})),\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{set2}\\ \quad (\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})) \to \mathsf{get\_nat}(\mathsf{A}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}}).\end{array}$$

And for the constructor function union we introduce two destructor functions left_set : set2 $\to$ set2 for the first argument of union and right_set : set2 $\to$ set2 for the second argument of union. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}\!:\!\mathsf{set2}\\ \quad \mathsf{A} \equiv \mathsf{union}(\mathsf{B},\mathsf{C}) \to \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})),\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{set2}\\ \quad (\forall\, \mathsf{B}, \mathsf{C}\!:\!\mathsf{set2}\ \mathsf{A} \not\equiv \mathsf{union}(\mathsf{B},\mathsf{C})) \to (\mathsf{left\_set}(\mathsf{A}) \equiv \mathsf{A} \wedge \mathsf{right\_set}(\mathsf{A}) \equiv \mathsf{A}),\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}\!:\!\mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B},\mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{empty} \wedge (\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})))\\ \qquad \to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true},\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}\!:\!\mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B},\mathsf{C}) \wedge (\mathsf{A} \equiv \mathsf{empty} \vee \exists\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{A} \equiv \mathsf{single}(\mathsf{x})))\\ \qquad \to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{false},\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}\!:\!\mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B},\mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{empty} \wedge (\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})) \wedge \Gamma_{\mathsf{union}}(\mathsf{B},\mathsf{C}) \equiv \mathsf{true})\\ \qquad \to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

They can be simplified to:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set2} \\ \quad \mathsf{A} \equiv \mathsf{union}(\mathsf{B}, \mathsf{C}) \rightarrow \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})),\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set2} \\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{empty} \wedge (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x}))) \\ \qquad \rightarrow \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true},\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set2} \\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B}, \mathsf{C}) \wedge (\mathsf{A} \equiv \mathsf{empty} \vee \exists\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \equiv \mathsf{single}(\mathsf{x}))) \\ \qquad \rightarrow \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{false}.\end{array}$$

Both reflexive destructor functions of the constructor function union, left_set and right_set, are 1-bounded, and their difference predicates $\Delta^1_{\mathsf{left\_set}} : \mathsf{set2} \rightarrow \mathsf{bool}$ and $\Delta^1_{\mathsf{right\_set}} : \mathsf{set2} \rightarrow \mathsf{bool}$, are defined by

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2} \\ \quad \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true} \\ \qquad \leftrightarrow \left(\begin{array}{l}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A} : \mathsf{set2} \\ \quad \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true} \\ \qquad \leftrightarrow \left(\begin{array}{l}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

For the data type set2 we will give constructive function and predicate specifications for delete, ins, inter, diff, card, $<_{\mathsf{set2}}$, $\leq_{\mathsf{set2}}$, $>_{\mathsf{set2}}$, and $\geq_{\mathsf{set2}}$.

9.1 delete : nat × set2 → set2

delete computes the delete operation on sets, thus it removes a specified object in a set, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set2} \\ \quad \mathsf{A} \equiv \mathsf{empty} \rightarrow \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{empty}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set2} \\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A})) \\ \qquad \rightarrow \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{empty}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set2} \\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A})) \\ \qquad \rightarrow \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{set2} \\ \quad \left(\begin{array}{l}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right) \\ \qquad \rightarrow \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{union}(\mathsf{delete}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A})))\end{array}$$

The recursion ordering of delete is well-founded. There is only one definition case with two recursive calls of delete. We abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)$$

by φ, and for the first recursive call we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{left\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set2} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right) \\ \quad\quad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set2} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right) \\ \quad\quad \rightarrow \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge \\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right).\end{array}$$

A proof of this property is quite simple using the definition of the destructor functions, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}{:}\,\mathsf{set2} \\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{empty} \wedge (\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x}))) \\ \quad\quad \rightarrow \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

For the second recursive call we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\,\mathsf{set2} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right) \\ \quad\quad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set2}\\ \quad\left(\begin{array}{l}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \left(\begin{array}{l}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right).\end{array}$$

Again, we can prove this obligation easily using

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}\!:\!\mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{empty} \wedge (\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})))\\ \qquad \to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

In addition, delete is a 2-bounded function symbol. To prove this property, first of all, we need to show that delete is completely specified, i.e.,

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set}\\ \quad \mathsf{A} \equiv \mathsf{empty}\vee\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A}))\vee\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A}))\vee\\ \quad \left(\begin{array}{l}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\end{array}$$

Then, we examine each definition case separately. For the first case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{empty}$$

by φ, and we obtain

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{empty}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the second definition case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A}))$$

by φ, and we obtain

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})), \mathsf{false}\rangle}\ \text{Equivalence}}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the third definition case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A}))$$

by φ, and we obtain

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{single}(\mathsf{get\_nat}(A)) \preceq_{\mathsf{set2}} \mathsf{single}(\mathsf{get\_nat}(A)), \mathsf{false}\rangle}\;\text{Identity}}{\langle \varphi, \mathsf{single}(\mathsf{get\_nat}(A)) \preceq_{\mathsf{set2}} A, \mathsf{false}\rangle}\;\text{Equation 1}$$

For the fourth definition case we abbreviate the invariant case condition

$$\left(\begin{array}{l} A \equiv \mathsf{union}(\mathsf{left\_set}(A), \mathsf{right\_set}(A)) \wedge \\ A \not\equiv \mathsf{empty} \wedge A \not\equiv \mathsf{single}(\mathsf{get\_nat}(A)) \end{array}\right)$$

by φ. Since this case is recursive, we may assume additional inference rules as induction hypotheses:

$$\xi_1 \Rightarrow \quad \cfrac{\langle \varphi, \mathsf{left\_set}(A) \preceq_{\mathsf{set2}} A, \Delta_1\rangle}{\langle \varphi, \mathsf{delete}(x, \mathsf{left\_set}(A)) \preceq_{\mathsf{set2}} \mathsf{left\_set}(A), \Delta^2_{\mathsf{delete}}(x, \mathsf{left\_set}(A)) \equiv \mathsf{true}\rangle}\;\text{Induction Hypothesis}$$

where ξ_1 is an abbreviation for the formula

$\forall\, x{:}\mathsf{nat}\ \forall\, A{:}\mathsf{set2}\ \varphi \rightarrow \Delta_1$

and

$$\xi_2 \Rightarrow \quad \cfrac{\langle \varphi, \mathsf{right\_set}(A) \preceq_{\mathsf{set2}} A, \Delta_2\rangle}{\langle \varphi, \mathsf{delete}(x, \mathsf{right\_set}(A)) \preceq_{\mathsf{set2}} \mathsf{right\_set}(A), \Delta^2_{\mathsf{delete}}(x, \mathsf{right\_set}(A)) \equiv \mathsf{true}\rangle}\;\text{Induction Hypothesis}$$

where ξ_2 is an abbreviation for the formula

$\forall\, x{:}\mathsf{nat}\ \forall\, A{:}\mathsf{set2}\ \varphi \rightarrow \Delta_2$

Using these additional rules, we obtain:

$$\cfrac{\cfrac{\cfrac{-}{\langle \varphi, A \preceq_{\mathsf{set2}} A, \mathsf{false}\rangle}\;\text{Identity}}{\langle \varphi, \mathsf{left\_set}(A) \preceq_{\mathsf{set2}} A, \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(A) \equiv \mathsf{true}\rangle}\;\text{Estimation}}{\langle \varphi, \mathsf{delete}(x, \mathsf{left\_set}(A)) \preceq_{\mathsf{set2}} \mathsf{left\_set}(A), \Delta^2_{\mathsf{delete}}(x, \mathsf{left\_set}(A)) \equiv \mathsf{true}\rangle}\;\text{Induction Hypothesis}$$

where to enable the application of the induction hypothesis, the formula

$\forall\, x{:}\mathsf{nat}\ \forall\, A{:}\mathsf{set2}\ \varphi \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(A) \equiv \mathsf{true})$

needs to be proved. On the other hand, we can derive:

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A})) \preceq_{\mathsf{set2}} \mathsf{right\_set}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where to allow the application of the induction hypothesis, the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set2}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be shown. Hence, we can derive:

$$\dfrac{\dfrac{\begin{array}{c}\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})) \preceq_{\mathsf{set2}} \mathsf{left\_set}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})) \equiv \mathsf{true}\rangle\,, \\ \langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A})) \preceq_{\mathsf{set2}} \mathsf{right\_set}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})) \equiv \mathsf{true}\rangle\end{array}}{\left\langle \begin{array}{c} \varphi, \mathsf{union}(\mathsf{delete}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A}))) \\ \preceq_{\mathsf{set2}} \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})), \\ \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{union}}(\mathsf{delete}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{union}(\mathsf{delete}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A}))) \preceq_{\mathsf{set2}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{union}}(\mathsf{delete}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where to enable the application of the Weak Embedding Rule, the formula

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set2} \\ \quad \varphi \to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{array}$$

has to be proved.

The corresponding difference predicate, $\Delta^2_{\mathsf{delete}} : \mathsf{nat} \times \mathsf{set2} \to \mathsf{bool}$, is now synthesized with the simplified difference formulas from the derivations in the Estimation Calculus as:

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set2} \\ \quad \mathsf{A} \equiv \mathsf{empty} \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set2} \\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \equiv \mathsf{get\_nat}(\mathsf{A})) \\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set2} \\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{x} \not\equiv \mathsf{get\_nat}(\mathsf{A})) \\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{set2}\\ \quad \left(\begin{array}{l}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \left(\begin{array}{l}\Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})) \equiv \mathsf{true}\vee\\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\vee\\ \Gamma_{\mathsf{union}}(\mathsf{delete}(\mathsf{x}, \mathsf{left\_set}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_set}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\end{array}\right)\end{array}$$

9.2 ins : nat × set2 → set2

ins computes the insertion operation of an element into a set, defined by:

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{set2}\\ \quad \mathsf{ins}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{union}(\mathsf{single}(\mathsf{x}), \mathsf{A})\end{array}$$

Since this a non-recursive constructive specification, we are done.

9.3 inter : set2 × set2 → set2

inter computes the intersection of two sets, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\colon \mathsf{set2}\\ \quad \mathsf{A} \equiv \mathsf{empty} \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{empty}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\colon \mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{get\_nat}(\mathsf{A}) \in \mathsf{B})\\ \qquad \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{A}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\colon \mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{get\_nat}(\mathsf{A}) \notin \mathsf{B})\\ \qquad \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{empty}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}\colon \mathsf{set2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \mathsf{inter}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{union}(\mathsf{inter}(\mathsf{left\_set}(\mathsf{A}), \mathsf{B}), \mathsf{inter}(\mathsf{right\_set}(\mathsf{A}), \mathsf{B}))\end{array}$$

The recursion ordering of inter is well-founded. There is only a single recursive definition case with two recursive calls. We abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)$$

by φ, and for the first recursive call we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\,\text{Identity}}{\langle \varphi, \mathsf{left\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\,\text{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2} \\ \quad \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{array}\right) \\ \quad\quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2} \\ \quad \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{array}\right) \\ \quad\quad \to \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true} \end{array}\right).\end{array}$$

A proof of this property is quite simple using the definition of the destructor functions, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set2} \\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{empty} \wedge (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x}))) \\ \quad\quad \to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

Similarly, for the second recursive call of inter we obtain:

$$\frac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true} \right\rangle}\ \text{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2} \\ \quad \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{array}\right) \\ \quad\quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2} \\ \quad \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{array}\right) \\ \quad\quad \to \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true} \end{array}\right).\end{array}$$

In addition, inter denotes a 1-bounded function symbol. To prove this property, first of all, we need to show that the specification of inter is case-complete,by

$$\forall\ \mathsf{A},\mathsf{B}:\mathsf{set2}\quad \begin{array}{c} \mathsf{A} \equiv \mathsf{empty} \lor \\ (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \land \mathsf{get\_nat}(\mathsf{A}) \in \mathsf{B}) \lor \\ (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \land \mathsf{get\_nat}(\mathsf{A}) \notin \mathsf{B}) \lor \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \\ \mathsf{A} \not\equiv \mathsf{empty} \land \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{array} \right) \end{array}$$

Next, we examine each definition case separately. For the first definition case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{empty}$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{empty}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Equation\ 1}$$

For the second definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \land \\ \mathsf{get\_nat}(\mathsf{A}) \in \mathsf{B} \end{array} \right)$$

by φ, and, using the Estimation Calculus, we obtain:

$$\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Identity}$$

For the third definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \land \\ \mathsf{get\_nat}(\mathsf{A}) \notin \mathsf{B} \end{array} \right)$$

by φ, and, using the Estimation Calculus, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})), \mathsf{false} \rangle}\ \mathsf{Equivalence}}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Equation\ 1}$$

For the fourth definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \\ \mathsf{A} \not\equiv \mathsf{empty} \land \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{array} \right)$$

by φ. Since this is a recursive case, we may assume the following induction hypotheses as additional inference rules:

$$\xi_1 \Rrightarrow \quad \frac{\langle \varphi, \mathsf{left\_set(A)} \preceq_{\mathsf{set2}} \mathsf{A}, \Delta_1 \rangle}{\langle \varphi, \mathsf{inter(left\_set(A), B)} \preceq_{\mathsf{set2}} \mathsf{left\_set(A)}, \Delta^1_{\mathsf{inter}}(\mathsf{left\_set(A), B}) \equiv \mathsf{true} \rangle}\ \text{Induction Hypothesis}$$

where ξ_1 is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2}\ \varphi \to \Delta_1$$

and

$$\xi_2 \Rrightarrow \quad \frac{\langle \varphi, \mathsf{right\_set(A)} \preceq_{\mathsf{set2}} \mathsf{A}, \Delta_2 \rangle}{\langle \varphi, \mathsf{inter(right\_set(A), B)} \preceq_{\mathsf{set2}} \mathsf{right\_set(A)}, \Delta^1_{\mathsf{inter}}(\mathsf{right\_set(A), B}) \equiv \mathsf{true} \rangle}\ \text{Induction Hypothesis}$$

where ξ_2 is an abbreviation of the formula

$$\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2}\ \varphi \to \Delta_2$$

Thus, we obtain

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{False} \rangle}\ \text{Identity}}{\langle \varphi, \mathsf{left\_set(A)} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{False} \lor \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{inter(left\_set(A), B)} \preceq_{\mathsf{set2}} \mathsf{left\_set(A)}, \Delta^1_{\mathsf{inter}}(\mathsf{left\_set(A), B}) \equiv \mathsf{true} \rangle}\ \text{Induction Hypothesis}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2}\ \varphi \to (\mathsf{False} \lor \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved. On the other hand we obtain the derivation:

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{False} \rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_set(A)} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{False} \lor \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true} \right\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{inter(right\_set(A), B)} \preceq_{\mathsf{set2}} \mathsf{right\_set(A)}, \Delta^1_{\mathsf{inter}}(\mathsf{right\_set(A), B}) \equiv \mathsf{true} \rangle}\ \text{Induction Hypothesis}$$

where to allow the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2}\ \varphi \to (\mathsf{False} \lor \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be shown. With these two estimation formulas we can now derive:

$$\dfrac{\dfrac{\begin{array}{c}\langle \varphi, \mathsf{inter}(\mathsf{left\_set}(A), B) \preceq_{\mathsf{set2}} \mathsf{left\_set}(A), \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}(A), B) \equiv \mathsf{true}\rangle\,, \\ \langle \varphi, \mathsf{inter}(\mathsf{right\_set}(A), B) \preceq_{\mathsf{set2}} \mathsf{right\_set}(A), \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}(A), B) \equiv \mathsf{true}\rangle\end{array}}{\left\langle \begin{array}{c} \varphi, \mathsf{union}(\mathsf{inter}(\mathsf{left\_set}(A), B), \mathsf{inter}(\mathsf{right\_set}(A), B)) \\ \preceq_{\mathsf{set2}} \mathsf{union}(\mathsf{left\_set}(A), \mathsf{right\_set}(A)), \\ \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}(A), B) \equiv \mathsf{true} \lor \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}(A), B) \equiv \mathsf{true} \lor \\ \Gamma_{\mathsf{union}}(\mathsf{inter}(\mathsf{left\_set}(A), B), \mathsf{inter}(\mathsf{right\_set}(A), B)) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{union}(\mathsf{inter}(\mathsf{left\_set}(A), B), \mathsf{inter}(\mathsf{right\_set}(A), B)) \preceq_{\mathsf{set2}} A, \\ \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}(A), B) \equiv \mathsf{true} \lor \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}(A), B) \equiv \mathsf{true} \lor \\ \Gamma_{\mathsf{union}}(\mathsf{inter}(\mathsf{left\_set}(A), B), \mathsf{inter}(\mathsf{right\_set}(A), B)) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where in order to enable the application of the Weak Embedding Rule, the formula

$$\forall\, A, B : \mathsf{set2}\ \varphi \to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(A), \mathsf{right\_set}(A)) \equiv \mathsf{true}$$

needs to be shown.

Now, we have proved that inter is 1-bounded, and the respective difference predicate $\Delta^1_{\mathsf{inter}} : \mathsf{set2} \times \mathsf{set2} \to \mathsf{bool}$ is synthesized using the simplified difference formulas from each call of the Estimation Calculus:

$$\begin{array}{l} \forall\, A, B : \mathsf{set2} \\ \quad A \equiv \mathsf{empty} \to \Delta^1_{\mathsf{inter}}(A, B) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, A, B : \mathsf{set2} \\ \quad \left(\begin{array}{l} A \equiv \mathsf{single}(\mathsf{get\_nat}(A)) \land \\ \qquad \mathsf{get\_nat}(A) \in B \end{array} \right) \\ \qquad \to \Delta^1_{\mathsf{inter}}(A, B) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, A, B : \mathsf{set2} \\ \quad \left(\begin{array}{l} A \equiv \mathsf{single}(\mathsf{get\_nat}(A)) \land \\ \qquad \mathsf{get\_nat}(A) \notin B \end{array} \right) \\ \qquad \to \Delta^1_{\mathsf{inter}}(A, B) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, A, B : \mathsf{set2} \\ \quad \left(\begin{array}{l} A \equiv \mathsf{union}(\mathsf{left\_set}(A), \mathsf{right\_set}(A)) \\ A \not\equiv \mathsf{empty} \land A \not\equiv \mathsf{single}(\mathsf{get\_nat}(A)) \end{array} \right) \\ \qquad \to \left(\begin{array}{l} \Delta^1_{\mathsf{inter}}(A, B) \equiv \mathsf{true} \\ \qquad \leftrightarrow \left(\begin{array}{c} \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}(A), B) \equiv \mathsf{true} \lor \\ \Delta^1_{\mathsf{inter}}(\mathsf{right\_set}(A), B) \equiv \mathsf{true} \lor \\ \Gamma_{\mathsf{union}}(\mathsf{inter}(\mathsf{left\_set}(A), B), \mathsf{inter}(\mathsf{right\_set}(A), B)) \equiv \mathsf{false} \end{array} \right) \end{array} \right) \end{array}$$

9.4 diff : set2 × set2 → set2

diff computes the difference of two sets, and it is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}\\ \quad \mathsf{A} \equiv \mathsf{empty} \to \mathsf{diff}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{empty}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{get\_nat}(\mathsf{A}) \in \mathsf{B})\\ \qquad \to \mathsf{diff}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{empty}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{get\_nat}(\mathsf{A}) \notin \mathsf{B})\\ \qquad \to \mathsf{diff}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{A}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \mathsf{diff}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{union}(\mathsf{diff}(\mathsf{left\_set}(\mathsf{A}), \mathsf{B}), \mathsf{diff}(\mathsf{right\_set}(\mathsf{A}), \mathsf{B}))\end{array}$$

The recursion ordering of diff is well-founded. There is only a single recursive definition case with two recursive calls. We abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)$$

by φ, and for the first recursive call we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{left\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\\ \qquad \to \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge\\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right).\end{array}$$

A proof of this property is quite simple using the definition of the destructor functions, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C} : \mathsf{set2}\\ \quad (\mathsf{A} \equiv \mathsf{union}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{empty} \wedge (\forall\, \mathsf{x} : \mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})))\\ \qquad \to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

Similarly, for the second recursive call of diff we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right) \\ \quad\ \to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right) \\ \quad\ \to \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right).\end{array}$$

In addition, diff denotes a 1-bounded function symbol. To prove this property, first of all, we need to show that the specification of diff is case-complete,by

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2} \\ \begin{array}{c}\mathsf{A} \equiv \mathsf{empty} \vee \\ (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{get\_nat}(\mathsf{A}) \in \mathsf{B}) \vee \\ (\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \mathsf{get\_nat}(\mathsf{A}) \notin \mathsf{B}) \vee \\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{array}\right)\end{array}\end{array}$$

Next, we examine each definition case separately. For the first definition case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{empty}$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{empty}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Equation 1}$$

For the second definition case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}(\mathsf{A}) \in \mathsf{B} \end{pmatrix}$$

by φ, and, using the Estimation Calculus, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})), \mathsf{false}\rangle}\ \mathsf{Equivalence}}{\langle \varphi, \mathsf{empty} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Equation\ 1}$$

For the third definition case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}(\mathsf{A}) \notin \mathsf{B} \end{pmatrix}$$

by φ, and, using the Estimation Calculus, we obtain:

$$\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}$$

For the fourth definition case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{pmatrix}$$

by φ. Since this is a recursive case, we may assume the following induction hypotheses as additional inference rules:

$$\xi_1 \Rightarrow \dfrac{\langle \varphi, \mathsf{left\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \Delta_1\rangle}{\langle \varphi, \mathsf{diff}(\mathsf{left\_set}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set2}} \mathsf{left\_set}(\mathsf{A}), \Delta^1_{\mathsf{diff}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\ \mathsf{Induction\ Hypothesis}$$

where ξ is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}\ \varphi \to \Delta_1$$

and

$$\xi_2 \Rightarrow \dfrac{\langle \varphi, \mathsf{right\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \Delta_2\rangle}{\langle \varphi, \mathsf{diff}(\mathsf{right\_set}(\mathsf{A}), \mathsf{B}) \preceq_{\mathsf{set2}} \mathsf{right\_set}(\mathsf{A}), \Delta^1_{\mathsf{diff}}(\mathsf{right\_set}(\mathsf{A}), \mathsf{B}) \equiv \mathsf{true}\rangle}\ \mathsf{Induction\ Hypothesis}$$

where ξ_2 is an abbreviation for the formula

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{set2}\ \varphi \to \Delta_2$$

Thus, we obtain

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{False}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{left\_set(A)} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{False} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{diff(left\_set(A), B)} \preceq_{\mathsf{set2}} \mathsf{left\_set(A)}, \Delta^1_{\mathsf{diff}}(\mathsf{left\_set(A), B}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B}:\mathsf{set2}\ \varphi \to (\mathsf{False} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved. On the other hand we obtain:

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{False}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_set(A)} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{False} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{diff(right\_set(A), B)} \preceq_{\mathsf{set2}} \mathsf{right\_set(A)}, \Delta^1_{\mathsf{diff}}(\mathsf{right\_set(A), B}) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where in order to allow the application of the induction hypothesis, the formula

$$\forall\, \mathsf{A}, \mathsf{B}:\mathsf{set2}\ \varphi \to (\mathsf{False} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true})$$

needs to be shown. Having derived the above estimation formulas we can now derive:

$$\dfrac{\dfrac{\begin{array}{c}\langle \varphi, \mathsf{diff(left\_set(A), B)} \preceq_{\mathsf{set2}} \mathsf{left\_set(A)}, \Delta^1_{\mathsf{diff}}(\mathsf{left\_set(A), B}) \equiv \mathsf{true}\rangle\,, \\ \langle \varphi, \mathsf{diff(right\_set(A), B)} \preceq_{\mathsf{set2}} \mathsf{right\_set(A)}, \Delta^1_{\mathsf{diff}}(\mathsf{right\_set(A), B}) \equiv \mathsf{true}\rangle\end{array}}{\left\langle \begin{array}{c}\varphi, \mathsf{union(diff(left\_set(A), B), diff(right\_set(A), B))} \\ \preceq_{\mathsf{set2}} \mathsf{union(left\_set(A), right\_set(A))}, \\ \Delta^1_{\mathsf{diff}}(\mathsf{left\_set(A), B}) \equiv \mathsf{true} \vee \Delta^1_{\mathsf{diff}}(\mathsf{right\_set(A), B}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{union}}(\mathsf{diff(left\_set(A), B), diff(right\_set(A), B)}) \equiv \mathsf{false}\end{array}\right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c}\varphi, \mathsf{union(diff(left\_set(A), B), diff(right\_set(A), B))} \preceq_{\mathsf{set2}} \mathsf{A}, \\ \Delta^1_{\mathsf{diff}}(\mathsf{left\_set(A), B}) \equiv \mathsf{true} \vee \Delta^1_{\mathsf{diff}}(\mathsf{right\_set(A), B}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{union}}(\mathsf{diff(left\_set(A), B), diff(right\_set(A), B)}) \equiv \mathsf{false}\end{array}\right\rangle}\ \text{Equation 3}$$

where to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}, \mathsf{B}:\mathsf{set2}\ \varphi \to \Gamma_{\mathsf{union}}(\mathsf{left\_set(A), right\_set(A)}) \equiv \mathsf{true}$$

has to be shown.

Now, we have proved that diff is 1-bounded, and the respective difference predicate Δ^1_{diff} : set2 × set2 → bool is synthesized using the simplified difference formulas from each call of the Estimation Calculus:

$$\forall\, \mathsf{A}, \mathsf{B}: \mathsf{set2}$$
$$\mathsf{A} \equiv \mathsf{empty} \rightarrow \Delta^1_{\mathsf{diff}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{false}$$

$$\forall\, \mathsf{A}, \mathsf{B}: \mathsf{set2}$$
$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}(\mathsf{A}) \in \mathsf{B} \end{pmatrix} \rightarrow \Delta^1_{\mathsf{diff}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{false}$$

$$\forall\, \mathsf{A}, \mathsf{B}: \mathsf{set2}$$
$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}(\mathsf{A}) \notin \mathsf{B} \end{pmatrix} \rightarrow \Delta^1_{\mathsf{diff}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{false}$$

$$\forall\, \mathsf{A}, \mathsf{B}: \mathsf{set2}$$
$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{pmatrix}$$
$$\rightarrow \begin{pmatrix} \Delta^1_{\mathsf{diff}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \\ \leftrightarrow \begin{pmatrix} \Delta^1_{\mathsf{diff}}(\mathsf{left\_set}(\mathsf{A}),\mathsf{B}) \equiv \mathsf{true} \vee \\ \Delta^1_{\mathsf{diff}}(\mathsf{right\_set}(\mathsf{A}),\mathsf{B}) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{union}}(\mathsf{diff}(\mathsf{left\_set}(\mathsf{A}),\mathsf{B}), \mathsf{diff}(\mathsf{right\_set}(\mathsf{A}),\mathsf{B})) \equiv \mathsf{false} \end{pmatrix} \end{pmatrix}$$

9.5 card : set2 → nat

card computes the cardinality of a set, and it is defined by:

$$\forall\, \mathsf{A}: \mathsf{set2}$$
$$\mathsf{A} \equiv \mathsf{empty} \rightarrow \mathsf{card}(\mathsf{A}) \equiv 0$$

$$\forall\, \mathsf{A}: \mathsf{set2}$$
$$\mathsf{A} \equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \rightarrow \mathsf{card}(\mathsf{A}) \equiv \mathsf{succ}(0)$$

$$\forall\, \mathsf{A}: \mathsf{set2}$$
$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{pmatrix}$$
$$\rightarrow \mathsf{card}(\mathsf{A}) \equiv \begin{pmatrix} (\mathsf{card}(\mathsf{left\_set}(\mathsf{A})) + \mathsf{card}(\mathsf{right\_set}(\mathsf{A}))) \\ -\mathsf{card}(\mathsf{inter}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))) \end{pmatrix}$$

The recursion ordering of card is well-founded. There is only one definition case with three recursive calls of card. Using the Estimation Calculus, abbreviating the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{pmatrix}$$

by φ, we obtain the following derivation for the first recursive call:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\;\mathsf{Identity}}{\left\langle \varphi, \mathsf{left\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\;\mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set2}\\ \quad\begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{pmatrix}\\ \quad\to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set2}\\ \quad\begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{pmatrix}\\ \quad\to \begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{pmatrix}.\end{array}$$

A proof of this property is quite simple using the definition of the destructor functions, i.e.,

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}{:}\mathsf{set2}\\ \quad(\mathsf{A} \equiv \mathsf{union}(\mathsf{B}, \mathsf{C}) \wedge \mathsf{A} \not\equiv \mathsf{empty} \wedge (\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \mathsf{A} \not\equiv \mathsf{single}(\mathsf{x})))\\ \qquad\to \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}.\end{array}$$

Similarly, for the second recursive call of card we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\;\mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\;\mathsf{Estimation}$$

To ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set2}\\ \quad\begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{pmatrix}\\ \quad\to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_set}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which simplifies to

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{set2}\\ \quad\begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A}))\end{pmatrix}\\ \quad\to \begin{pmatrix} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A}))\wedge\\ \Gamma_{\mathsf{union}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true}\end{pmatrix}.\end{array}$$

And for the third recursive call we obtain the derivation

$$\cfrac{\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{left\_set}(\mathsf{A}) \preceq_{\mathsf{set2}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}}{\left\langle \begin{array}{c} \varphi, \mathsf{inter}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \preceq_{\mathsf{set2}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true} \vee \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle}\ \mathsf{Estimation}$$

In order to prove the strict relation, we need to show

$$\begin{array}{l} \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{set2} \\ \quad \left(\begin{array}{l} \mathsf{A} \equiv \mathsf{union}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \wedge \\ \mathsf{A} \not\equiv \mathsf{empty} \wedge \mathsf{A} \not\equiv \mathsf{single}(\mathsf{get\_nat}(\mathsf{A})) \end{array} \right) \\ \quad \to \left(\begin{array}{c} \mathsf{false} \vee \Delta^1_{\mathsf{left\_set}}(\mathsf{A}) \equiv \mathsf{true} \vee \\ \Delta^1_{\mathsf{inter}}(\mathsf{left\_set}(\mathsf{A}), \mathsf{right\_set}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right) \end{array}$$

9.6 $<_{\mathsf{set2}}$: set2 × set2 → bool

$<_{\mathsf{set2}}$ computes the less-than-relation on sets, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2} \\ \quad (\mathsf{A} <_{\mathsf{set2}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{card}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{card}(\mathsf{B})) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive constructive definition, we are done. However note, that $<_{\mathsf{set2}}$ denotes a well-founded order relation.

9.7 $\leq_{\mathsf{set2}}$: set2 × set2 → bool

$\leq_{\mathsf{set2}}$ computes the less-than-or-equal-relation on sets, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2} \\ \quad (\mathsf{A} \leq_{\mathsf{set2}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{card}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{card}(\mathsf{B})) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive constructive definition, we are done.

9.8 $>_{\mathsf{set2}}$: set2 × set2 → bool

$>_{\mathsf{set2}}$ computes the greater-than-relation on sets, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{set2} \\ \quad (\mathsf{A} >_{\mathsf{set2}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} <_{\mathsf{set2}} \mathsf{A}) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive constructive definition, we are done.

9.9 $\geq_{\mathsf{set2}}$: $\mathsf{set2} \times \mathsf{set2} \rightarrow \mathsf{bool}$

$\geq_{\mathsf{set2}}$ computes the greater-than-or-equal-relation on sets, and it is defined by:

$$\begin{array}{l}\forall\ \mathsf{A}, \mathsf{B} : \mathsf{set2} \\ \quad (\mathsf{A} \geq_{\mathsf{set2}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} \leq_{\mathsf{set2}} \mathsf{A}) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive constructive definition, we are done.

10

Binary Words, binword

This specification of binary words, binword, uses four constructor functions $0 :\to \mathsf{binword}$, generating the binary word for zero, $1 :\to \mathsf{binword}$, generating the binary word for one, $\mathsf{succ0} : \mathsf{binword} \to \mathsf{binword}$, adding a zero to the end of a binary word, and $\mathsf{succ1} : \mathsf{binword} \to \mathsf{binword}$, adding a one to the end of a binary word. Equality on binword is specified by the axioms:

$0 \not\equiv 1$

$\forall\, x : \mathsf{binword}$
  $0 \equiv \mathsf{succ0}(x) \leftrightarrow x \equiv 0$

$\forall\, x : \mathsf{binword}$
  $0 \not\equiv \mathsf{succ1}(x)$

$\forall\, x : \mathsf{binword}$
  $1 \not\equiv \mathsf{succ0}(x)$

$\forall\, x : \mathsf{binword}$
  $1 \equiv \mathsf{succ1}(x) \leftrightarrow x \equiv 0$

$\forall\, x, y : \mathsf{binword}$
  $\mathsf{succ0}(x) \not\equiv \mathsf{succ1}(y)$

$\forall\, x, y : \mathsf{binword}$
  $\mathsf{succ0}(x) \equiv \mathsf{succ0}(y) \leftrightarrow x \equiv y$

$\forall\, x, y : \mathsf{binword}$
  $\mathsf{succ1}(x) \equiv \mathsf{succ1}(y) \leftrightarrow x \equiv y$

By the above specification we have defined a non-freely generated data type. Hence, we must prove the constructor functions succ0 and succ1 to be size increasing by using the respective implementation specification. Furthermore, the strictness predicates $\Theta^1_{\mathsf{succ0}} : \mathsf{binword} \to \mathsf{bool}$ and $\Theta^1_{\mathsf{succ1}} : \mathsf{binword} \to \mathsf{bool}$, as well as the minimal representation predicates $\Gamma_{\mathsf{succ0}} : \mathsf{binword} \to \mathsf{bool}$ and $\Gamma_{\mathsf{succ1}} : \mathsf{binword} \to \mathsf{bool}$ have to be synthesized.

The implementation specification is automatically generated using the constructor functions $0_I :\to \mathsf{binword}_I$, $1_I :\to \mathsf{binword}_I$, $\mathsf{succ0}_I : \mathsf{binword}_I \to \mathsf{binword}_I$, $\mathsf{succ1}_I : \mathsf{binword}_I \to \mathsf{binword}_I$, and the new equality predicate $\mathsf{Eq}_{\mathsf{binword}_I} : \mathsf{binword}_I \times \mathsf{binword}_I \to \mathsf{bool}$.

$0_I \not\equiv 1_I$

$\forall\, x : \mathsf{binword}_I$
  $0_I \not\equiv \mathsf{succ0}_I(x)$

$\forall\, x : \mathsf{binword}_I$
  $0_I \not\equiv \mathsf{succ1}_I(x)$

$\forall\, x : \mathsf{binword}_I$
  $1_I \not\equiv \mathsf{succ0}_I(x)$

$\forall\, x : \mathsf{binword}_I$
  $1_I \not\equiv \mathsf{succ1}_I(x)$

$\forall\, x, y : \mathsf{binword}_I$
  $\mathsf{succ0}_I(x) \not\equiv \mathsf{succ1}_I(y)$

$\forall\, x, y : \mathsf{binword}_I$
  $\mathsf{succ0}_I(x) \equiv \mathsf{succ0}_I(y) \leftrightarrow x \equiv y$

$\forall\, x, y : \mathsf{binword}_I$
  $\mathsf{succ1}_I(x) \equiv \mathsf{succ1}_I(y) \leftrightarrow x \equiv y$

$\mathsf{Eq}_{\mathsf{binword}_I}(0_I, 1_I) \equiv \mathsf{false}$

$\forall\, x : \mathsf{binword}_I$
  $\mathsf{Eq}_{\mathsf{binword}_I}(0_I, \mathsf{succ0}_I(x)) \equiv \mathsf{true} \leftrightarrow \mathsf{Eq}_{\mathsf{binword}_I}(x, 0_I) \equiv \mathsf{true}$

$\forall\, x : \mathsf{binword}_I$
  $\mathsf{Eq}_{\mathsf{binword}_I}(0_I, \mathsf{succ1}_I(x)) \equiv \mathsf{false}$

$\forall\, x : \mathsf{binword}_I$
  $\mathsf{Eq}_{\mathsf{binword}_I}(1_I, \mathsf{succ0}_I(x)) \equiv \mathsf{false}$

$\forall\, x : \mathsf{binword}_I$
  $\mathsf{Eq}_{\mathsf{binword}_I}(1_I, \mathsf{succ1}_I(x)) \equiv \mathsf{true} \leftrightarrow \mathsf{Eq}_{\mathsf{binword}_I}(x, 0_I) \equiv \mathsf{true}$

$\forall\, x, y : \mathsf{binword}_I$
  $\mathsf{Eq}_{\mathsf{binword}_I}(\mathsf{succ0}_I(x), \mathsf{succ1}_I(y)) \equiv \mathsf{false}$

$\forall\, x, y : \mathsf{binword}_I$
  $\mathsf{Eq}_{\mathsf{binword}_I}(\mathsf{succ0}_I(x), \mathsf{succ0}_I(y)) \equiv \mathsf{true} \leftrightarrow \mathsf{Eq}_{\mathsf{binword}_I}(x, y) \equiv \mathsf{true}$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{succ1}_{\mathrm{I}}(\mathsf{x}), \mathsf{succ1}_{\mathrm{I}}(\mathsf{y})) \equiv \mathsf{true} \leftrightarrow \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}, \mathsf{y}) \equiv \mathsf{true}$

$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}, \mathsf{x}) \equiv \mathsf{true}$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}_{\mathrm{I}}$
$\quad \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}, \mathsf{y}) \equiv \mathsf{true} \rightarrow \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{y}, \mathsf{x}) \equiv \mathsf{true}$

$\forall\, \mathsf{x}, \mathsf{y}, \mathsf{z} : \mathsf{binword}_{\mathrm{I}}$
$\quad (\mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}, \mathsf{y}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{y}, \mathsf{z}) \equiv \mathsf{true})$
$\quad\quad \rightarrow \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}, \mathsf{z}) \equiv \mathsf{true}$

Since $\mathsf{binword}_{\mathrm{I}}$ is freely generated, the strictness predicates $\theta^1_{\mathsf{succ0}_{\mathrm{I}}} : \mathsf{binword}_{\mathrm{I}} \rightarrow \mathsf{bool}$ and $\theta^1_{\mathsf{succ1}_{\mathrm{I}}} : \mathsf{binword}_{\mathrm{I}} \rightarrow \mathsf{bool}$, as well as the minimal representation predicates $\gamma_{\mathsf{succ0}_{\mathrm{I}}} : \mathsf{binword}_{\mathrm{I}} \rightarrow \mathsf{bool}$ and $\gamma_{\mathsf{succ0}_{\mathrm{I}}} : \mathsf{binword}_{\mathrm{I}} \rightarrow \mathsf{bool}$ are defined by:

$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}}$
$\quad \theta^1_{\mathsf{succ0}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true}$

$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}}$
$\quad \theta^1_{\mathsf{succ1}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true}$

$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}}$
$\quad \gamma_{\mathsf{succ0}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true}$

$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}}$
$\quad \gamma_{\mathsf{succ1}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true}$

In addition, all constructor functions of $\mathsf{binword}_{\mathrm{I}}$ are non-overlapping. Hence, the destructor function $\mathsf{pred0}_{\mathrm{I}} : \mathsf{binword}_{\mathrm{I}} \rightarrow \mathsf{binword}_{\mathrm{I}}$ for the constructor function $\mathsf{succ0}$ and the destructor function $\mathsf{pred1}_{\mathrm{I}} : \mathsf{binword}_{\mathrm{I}} \rightarrow \mathsf{binword}_{\mathrm{I}}$ for the constructor function $\mathsf{succ1}_{\mathrm{I}}$ are defined by:

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}_{\mathrm{I}}$
$\quad \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{y}) \rightarrow \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}))$

$\mathsf{pred0}_{\mathrm{I}}(\mathsf{0}_{\mathrm{I}}) \equiv \mathsf{0}_{\mathrm{I}}$

$\mathsf{pred0}_{\mathrm{I}}(\mathsf{1}_{\mathrm{I}}) \equiv \mathsf{1}_{\mathrm{I}}$

$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}}$
$\quad \mathsf{pred0}_{\mathrm{I}}(\mathsf{succ1}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{x})$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}_{\mathrm{I}}$
$\quad \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{y}) \rightarrow \gamma_{\mathsf{succ0}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{true}$

$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}_{\mathrm{I}}$
$\quad \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{y}) \rightarrow \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}))$

$\mathsf{pred1}_{\mathrm{I}}(\mathsf{0}_{\mathrm{I}}) \equiv \mathsf{0}_{\mathrm{I}}$

$\mathsf{pred1}_{\mathrm{I}}(\mathsf{1}_{\mathrm{I}}) \equiv \mathsf{1}_{\mathrm{I}}$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \mathsf{pred1}_{\mathrm{I}}(\mathsf{succ0}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{x})$$

$$\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}_{\mathrm{I}} \quad \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{y}) \to \gamma_{\mathsf{succ1}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{true}$$

Now, $\mathsf{pred0}_{\mathrm{I}}$ and $\mathsf{pred1}_{\mathrm{I}}$ are both 1-bounded with difference predicates $\Delta^{\mathrm{I1}}_{\mathsf{pred0}_{\mathrm{I}}} : \mathsf{binword}_{\mathrm{I}} \to \mathsf{bool}$ and $\Delta^{\mathrm{I1}}_{\mathsf{pred1}_{\mathrm{I}}} : \mathsf{binword}_{\mathrm{I}} \to \mathsf{bool}$, defined by

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \Delta^{\mathrm{I1}}_{\mathsf{pred0}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}))$$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \Delta^{\mathrm{I1}}_{\mathsf{pred1}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}))$$

Furthermore, the function $\mathsf{term\_size}_{\mathsf{binword}_{\mathrm{I}}} : \mathsf{binword}_{\mathrm{I}} \to \mathsf{nat}$ is synthesized by:

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \mathsf{x} \equiv 0_{\mathrm{I}} \to \mathsf{term\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{A}) \equiv 0$$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \mathsf{x} \equiv 1_{\mathrm{I}} \to \mathsf{term\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{A}) \equiv 0$$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x})) \to \mathsf{term\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{term\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x})))$$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x})) \to \mathsf{term\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{term\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x})))$$

In order to have easier proofs, we specify a function $\mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}} : \mathsf{binword}_{\mathrm{I}} \to \mathsf{nat}$, by

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \mathsf{x} \equiv 0_{\mathrm{I}} \to \mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}) \equiv 0$$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \mathsf{x} \equiv 1_{\mathrm{I}} \to \mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}) \equiv 0$$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x})) \wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{true} \end{pmatrix} \to \mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}) \equiv 0$$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x})) \wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false} \end{pmatrix} \to \mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x})))$$

$$\forall\, \mathsf{x} : \mathsf{binword}_{\mathrm{I}} \quad \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x})) \wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{true} \end{pmatrix} \to \mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}) \equiv 0$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false} \end{array}\right) \\ \to \mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x})))$$

The specification of $\mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}$ is case-distinct, as proved by

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 0_{\mathrm{I}}\wedge \\ \mathsf{x} \equiv 1_{\mathrm{I}} \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 0_{\mathrm{I}}\wedge \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{true} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 0_{\mathrm{I}}\wedge \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 0_{\mathrm{I}}\wedge \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{true} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 0_{\mathrm{I}}\wedge \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 1_{\mathrm{I}}\wedge \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{true} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 1_{\mathrm{I}}\wedge \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 1_{\mathrm{I}}\wedge \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{true} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \neg\left(\begin{array}{c} \mathsf{x} \equiv 1_{\mathrm{I}}\wedge \\ \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}))\wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false} \end{array}\right) \end{array}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword_I}\ \neg\left(\begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0_I}(\mathsf{pred0_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred0_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{true} \end{pmatrix} \wedge \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0_I}(\mathsf{pred0_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred0_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{false} \end{pmatrix}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword_I}\ \neg\left(\begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0_I}(\mathsf{pred0_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred0_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{true} \end{pmatrix} \wedge \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ1_I}(\mathsf{pred1_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred1_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{true} \end{pmatrix}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword_I}\ \neg\left(\begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0_I}(\mathsf{pred0_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred0_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{true} \end{pmatrix} \wedge \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ1_I}(\mathsf{pred1_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred1_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{false} \end{pmatrix}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword_I}\ \neg\left(\begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0_I}(\mathsf{pred0_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred0_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{false} \end{pmatrix} \wedge \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ1_I}(\mathsf{pred1_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred1_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{true} \end{pmatrix}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword_I}\ \neg\left(\begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0_I}(\mathsf{pred0_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred0_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{false} \end{pmatrix} \wedge \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ1_I}(\mathsf{pred1_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred1_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{false} \end{pmatrix}\right)$$

$$\forall\, \mathsf{x}\colon \mathsf{binword_I}\ \neg\left(\begin{pmatrix} \mathsf{x} \equiv \mathsf{succ1_I}(\mathsf{pred1_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred1_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{true} \end{pmatrix} \wedge \begin{pmatrix} \mathsf{x} \equiv \mathsf{succ1_I}(\mathsf{pred1_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred1_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{false} \end{pmatrix}\right)$$

Furthermore, the recursion ordering of $\mathsf{min\_size_{binword_I}}$ is well-founded. To prove that, we use the Estimation Calculus. There are two recursive cases with a single recursive call in each. For the first recursive case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{x} \equiv \mathsf{succ0_I}(\mathsf{pred0_I}(\mathsf{x}))\wedge \\ \mathsf{Eq_{binword_I}}(\mathsf{pred0_I}(\mathsf{x}), \mathsf{0_I}) \equiv \mathsf{false} \end{pmatrix}$$

by φ. Thus, we obtain the following derivation in the Estimation Calculus:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{binword}_{\mathrm{I}}} \mathsf{x}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{pred0}_{\mathrm{I}}(\mathsf{x}) \preceq_{\mathsf{binword}_{\mathrm{I}}} \mathsf{x}, \mathsf{false} \vee \Delta^1_{\mathsf{pred0}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \quad \left(\begin{array}{c}\mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x})) \wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false}\end{array}\right) \\ \quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred0}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \quad \left(\begin{array}{c}\mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x})) \wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false}\end{array}\right) \\ \quad \to \mathsf{x} \equiv \mathsf{succ0}_{\mathrm{I}}(\mathsf{pred0}_{\mathrm{I}}(\mathsf{x}))\end{array}$$

For the second recursive case we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x})) \wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false}\end{array}\right)$$

by φ. Thus, we obtain the following derivation in the Estimation Calculus:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{binword}_{\mathrm{I}}} \mathsf{x}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{pred1}_{\mathrm{I}}(\mathsf{x}) \preceq_{\mathsf{binword}_{\mathrm{I}}} \mathsf{x}, \mathsf{false} \vee \Delta^1_{\mathsf{pred1}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \quad \left(\begin{array}{c}\mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x})) \wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false}\end{array}\right) \\ \quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred1}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}_{\mathrm{I}} \\ \quad \left(\begin{array}{c}\mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x})) \wedge \\ \mathsf{Eq}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}), 0_{\mathrm{I}}) \equiv \mathsf{false}\end{array}\right) \\ \quad \to \mathsf{x} \equiv \mathsf{succ1}_{\mathrm{I}}(\mathsf{pred1}_{\mathrm{I}}(\mathsf{x}))\end{array}$$

Now, we need to prove that the above axiomatization of $\mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}$ computes the minimal size of a binary word, indeed. Therefore we need to show the following proof obligations

$$\begin{aligned}&\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}_\mathrm{I}\\ &\quad \mathsf{Eq}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x},\mathsf{y}) \equiv \mathsf{true} \to (\mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}) \leq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{y})) \equiv \mathsf{true}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\ \exists\, \mathsf{y} : \mathsf{binword}_\mathrm{I}\\ &\quad \mathsf{Eq}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x},\mathsf{y}) \equiv \mathsf{true} \wedge (\mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}) \geq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{y})) \equiv \mathsf{true}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}_\mathrm{I}\\ &\quad \mathsf{Eq}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x},\mathsf{y}) \equiv \mathsf{true} \to \mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}) \equiv \mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{y})\end{aligned}$$

Next, we need to show that succ0 and succ1 denote size increasing constructor functions. To do that, we prove:

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad (\mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{succ0}(\mathsf{x}))) \equiv \mathsf{true}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad (\mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{succ1}(\mathsf{x}))) \equiv \mathsf{true}\end{aligned}$$

Finally, we need to define the strictness predicates $\Theta^1_{\mathsf{succ0}_\mathrm{I}} : \mathsf{binword}_\mathrm{I} \to \mathsf{bool}$ and $\Theta^1_{\mathsf{succ1}_\mathrm{I}} : \mathsf{binword}_\mathrm{I} \to \mathsf{bool}$, as well as the minimal representation predicates $\Gamma_{\mathsf{succ0}_\mathrm{I}} : \mathsf{binword}_\mathrm{I} \to \mathsf{bool}$ and $\Gamma_{\mathsf{succ1}_\mathrm{I}} : \mathsf{binword}_\mathrm{I} \to \mathsf{bool}$. We suggest the following definitions:

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad \Theta^1_{\mathsf{succ0}_\mathrm{I}}(\mathsf{x}) \equiv \mathsf{true}\\ &\qquad \leftrightarrow \mathsf{Eq}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}, 0_\mathrm{I}) \equiv \mathsf{false}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad \Theta^1_{\mathsf{succ1}_\mathrm{I}}(\mathsf{x}) \equiv \mathsf{true}\\ &\qquad \leftrightarrow \mathsf{Eq}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}, 0_\mathrm{I}) \equiv \mathsf{false}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad \Gamma_{\mathsf{succ0}_\mathrm{I}}(\mathsf{x}) \equiv \mathsf{true}\\ &\qquad \leftrightarrow \mathsf{Eq}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}, 0_\mathrm{I}) \equiv \mathsf{false}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad \Gamma_{\mathsf{succ1}_\mathrm{I}}(\mathsf{x}) \equiv \mathsf{true}\\ &\qquad \leftrightarrow \mathsf{Eq}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}, 0_\mathrm{I}) \equiv \mathsf{false}\end{aligned}$$

However, we have to prove that our suggestions really define the strictness predicates and the minimal representation predicate. Hence, we need to show that

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad \Theta^1_{\mathsf{succ0}_\mathrm{I}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow (\mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{succ0}_\mathrm{I}(\mathsf{x}))) \equiv \mathsf{true}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad \Theta^1_{\mathsf{succ1}_\mathrm{I}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow (\mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{succ1}_\mathrm{I}(\mathsf{x}))) \equiv \mathsf{true}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{x} : \mathsf{binword}_\mathrm{I}\\ &\quad \Gamma_{\mathsf{succ0}_\mathrm{I}}(\mathsf{x}) \equiv \mathsf{true}\\ &\qquad \leftrightarrow \mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{succ0}_\mathrm{I}(\mathsf{x})) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{binword}_\mathrm{I}}(\mathsf{x}))\end{aligned}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{binword}_{\mathrm{I}}\\ \quad \Gamma_{\mathsf{succ1}_{\mathrm{I}}}(\mathsf{x}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{succ0}_{\mathrm{I}}(\mathsf{x})) \equiv \mathsf{succ}(\mathsf{min\_size}_{\mathsf{binword}_{\mathrm{I}}}(\mathsf{x}))\end{array}$$

Having done so, we know for our original specification binword that the constructor functions succ0 and succ1 are size increasing, and we can translate the strictness predicates and the minimal representation predicates into the original specification. Hence, we obtain:

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{binword}\\ \quad \Theta^1_{\mathsf{succ0}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow \mathsf{x} \not\equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{binword}\\ \quad \Theta^1_{\mathsf{succ1}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow \mathsf{x} \not\equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{binword}\\ \quad \Gamma_{\mathsf{succ0}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow \mathsf{x} \not\equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{binword}\\ \quad \Gamma_{\mathsf{succ1}}(\mathsf{x}) \equiv \mathsf{true} \leftrightarrow \mathsf{x} \not\equiv 0\end{array}$$

The data type binword possesses overlapping constructor functions, since, for instance,

$$0 \equiv \mathsf{succ0}(0)$$

Thus, we cannot use the simplified construction scheme for the destructor functions.

The destructor function pred0 : binword → binword for the constructor function succ0 and the destructor function pred1 : binword → binword for the constructor function succ1 are defined by the following (already simplified) axioms:

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{binword}\\ \quad \mathsf{x} \equiv \mathsf{succ0}(\mathsf{y}) \rightarrow \mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x})),\end{array}$$

$$\mathsf{pred0}(0) \equiv 0$$

$$\mathsf{pred0}(1) \equiv 1$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{binword}\\ \quad \mathsf{pred0}(\mathsf{succ1}(\mathsf{x})) \equiv \mathsf{succ1}(\mathsf{x})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ0}(\mathsf{y}) \wedge \mathsf{x} \not\equiv 0)\\ \qquad \rightarrow \Gamma_{\mathsf{succ0}}(\mathsf{pred0}(\mathsf{x})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ0}(\mathsf{y}) \wedge \mathsf{x} \equiv 0)\\ \qquad \rightarrow \Gamma_{\mathsf{succ0}}(\mathsf{pred0}(\mathsf{x})) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{binword}\\ \quad \mathsf{x} \equiv \mathsf{succ1}(\mathsf{y}) \rightarrow \mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x})),\end{array}$$

$$\mathsf{pred1}(0) \equiv 0$$

$$\mathsf{pred1}(1) \equiv 1$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{binword}\\ \quad \mathsf{pred1}(\mathsf{succ0}(\mathsf{x})) \equiv \mathsf{succ0}(\mathsf{x})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}:\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ1}(\mathsf{y}) \wedge \mathsf{x} \not\equiv 1)\\ \qquad \rightarrow \Gamma_{\mathsf{succ1}}(\mathsf{pred1}(\mathsf{x})) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}:\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ1}(\mathsf{y}) \wedge \mathsf{x} \equiv 1)\\ \qquad \rightarrow \Gamma_{\mathsf{succ1}}(\mathsf{pred1}(\mathsf{x})) \equiv \mathsf{false}\end{array}$$

Both reflexive destructor functions pred0 and pred1, are 1-bounded, and their difference predicates $\Delta^1_{\mathsf{pred0}} : \mathsf{binword} \rightarrow \mathsf{bool}$ and $\Delta^1_{\mathsf{pred1}} : \mathsf{binword} \rightarrow \mathsf{bool}$, are defined by

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{binword}\\ \quad \Delta^1_{\mathsf{pred0}}(\mathsf{x}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x}))\wedge\\ \Gamma_{\mathsf{succ0}}(\mathsf{pred0}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{binword}\\ \quad \Delta^1_{\mathsf{pred1}}(\mathsf{x}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x}))\wedge\\ \Gamma_{\mathsf{succ1}}(\mathsf{pred1}(\mathsf{A})) \equiv \mathsf{true}\end{array}\right)\end{array}$$

For the data type binword we will give constructive function and predicate specifications for succ, pred, $+$, $-$, $<_{\mathsf{binword}}$, $\leq_{\mathsf{binword}}$, $>_{\mathsf{binword}}$, and $\geq_{\mathsf{binword}}$.

10.1 succ : binword → binword

succ computes the addition of 1 and the specified binary word, defined by:

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{binword}\\ \quad \mathsf{x} \equiv 0 \rightarrow \mathsf{succ}(\mathsf{x}) \equiv 1\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{binword}\\ \quad \mathsf{x} \equiv 1 \rightarrow \mathsf{succ}(\mathsf{x}) \equiv \mathsf{succ0}(\mathsf{x})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 0)\\ \qquad \rightarrow \mathsf{succ}(\mathsf{x}) \equiv \mathsf{succ1}(\mathsf{pred0}(\mathsf{x}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}:\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 1)\\ \qquad \rightarrow \mathsf{succ}(\mathsf{x}) \equiv \mathsf{succ0}(\mathsf{succ}(\mathsf{pred1}(\mathsf{x})))\end{array}$$

The recursion ordering of succ is well-founded. There is only a single recursive definition case with a single recursive call. We abbreviate the invariant case condition

$$(\mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 1)$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{binword}} \mathsf{x}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{pred1}(\mathsf{x}) \preceq_{\mathsf{binword}} \mathsf{x}, \mathsf{false} \vee \Delta^{1}_{\mathsf{pred1}}(\mathsf{x}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

To ensure the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 1)\\ \qquad \to (\mathsf{false} \vee \Delta^{1}_{\mathsf{pred1}}(\mathsf{x}) \equiv \mathsf{true})\end{array}$$

which can be easily proved using the definition of the involved functions.

10.2 pred : binword → binword

pred computes the subtraction by 1 from the specified binary word, defined by:

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}\\ \quad \mathsf{x} \equiv 0 \to \mathsf{pred}(\mathsf{x}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}\\ \quad \mathsf{x} \equiv 1 \to \mathsf{pred}(\mathsf{x}) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 0)\\ \qquad \to \mathsf{pred}(\mathsf{x}) \equiv \mathsf{succ1}(\mathsf{pred}(\mathsf{pred0}(\mathsf{x})))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 1)\\ \qquad \to \mathsf{pred}(\mathsf{x}) \equiv \mathsf{succ0}(\mathsf{pred1}(\mathsf{x}))\end{array}$$

The recursion ordering of pred is well-founded. There is only a single recursive definition case with a single recursive call. We abbreviate the invariant case condition

$$(\mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 0)$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{binword}} \mathsf{x}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{pred0}(\mathsf{x}) \preceq_{\mathsf{binword}} \mathsf{x}, \mathsf{false} \vee \Delta^{1}_{\mathsf{pred0}}(\mathsf{x}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

To ensure the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}{:}\,\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 0)\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred0}}(\mathsf{x}) \equiv \mathsf{true})\end{array}$$

which can be easily proved using the definition of the involved functions.

10.3 $+ : \mathsf{binword} \times \mathsf{binword} \to \mathsf{binword}$

$+$ computes the addition on binary words, defined by:

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}{:}\,\mathsf{binword}\\ \quad \mathsf{x} \equiv 0 \to (\mathsf{x}+\mathsf{y}) \equiv \mathsf{y}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}{:}\,\mathsf{binword}\\ \quad \mathsf{x} \equiv 1 \to (\mathsf{x}+\mathsf{y}) \equiv \mathsf{succ}(\mathsf{y})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}{:}\,\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 0)\\ \qquad \to (\mathsf{x}+\mathsf{y}) \equiv (\mathsf{pred0}(\mathsf{x}) + (\mathsf{pred0}(\mathsf{x}) + \mathsf{y}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}{:}\,\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 1)\\ \qquad \to (\mathsf{x}+\mathsf{y}) \equiv \mathsf{succ}(\mathsf{pred1}(\mathsf{x}) + (\mathsf{pred1}(\mathsf{x}) + \mathsf{y}))\end{array}$$

The recursion ordering of $+$ is well-founded. There are two recursive definition cases with two recursive calls in each, however, both recursive calls coincide in the first parameter. For the first recursive case we abbreviate the invariant case condition

$$(\mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 0)$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{binword}} \mathsf{x}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{pred0}(\mathsf{x}) \preceq_{\mathsf{binword}} \mathsf{x}, \mathsf{false} \vee \Delta^1_{\mathsf{pred0}}(\mathsf{x}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

To ensure the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}{:}\,\mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 0)\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred0}}(\mathsf{x}) \equiv \mathsf{true})\end{array}$$

which can be easily proved using the definition of the involved functions. For the second recursive case we abbreviate the invariant case condition

$$(\mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 1)$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{x} \preceq_{\mathsf{binword}} \mathsf{x}, \mathsf{false}\rangle}\;\text{Identity}}{\left\langle \varphi, \mathsf{pred1}(\mathsf{x}) \preceq_{\mathsf{binword}} \mathsf{x}, \mathsf{false} \vee \Delta^1_{\mathsf{pred1}}(\mathsf{x}) \equiv \mathsf{true}\right\rangle}\;\text{Estimation}$$

To ensure the strict relation, we have to prove

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}\\ \quad (\mathsf{x} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{x})) \wedge \mathsf{x} \not\equiv 1)\\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred1}}(\mathsf{x}) \equiv \mathsf{true})\end{array}$$

which can be easily proved using the definition of the involved functions.

10.4 $-$: binword $\times$ binword $\to$ binword

$-$ computes the subtraction on binary words, defined by:

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}\\ \quad \mathsf{y} \equiv 0 \to (\mathsf{x} - \mathsf{y}) \equiv \mathsf{x}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}\\ \quad \mathsf{y} \equiv 1 \to (\mathsf{x} - \mathsf{y}) \equiv \mathsf{pred}(\mathsf{x})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}\\ \quad (\mathsf{y} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{y})) \wedge \mathsf{y} \not\equiv 0)\\ \qquad \to (\mathsf{x} - \mathsf{y}) \equiv ((\mathsf{x} - \mathsf{pred0}(\mathsf{y})) - \mathsf{pred0}(\mathsf{y}))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword}\\ \quad (\mathsf{y} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{y})) \wedge \mathsf{y} \not\equiv 1)\\ \qquad \to (\mathsf{x} - \mathsf{y}) \equiv \mathsf{pred}((\mathsf{x} - \mathsf{pred1}(\mathsf{y})) - \mathsf{pred1}(\mathsf{y}))\end{array}$$

The recursion ordering of $-$ is well-founded. There are two recursive definition cases with two recursive calls in each, however, both recursive calls coincide in the first parameter. For the first recursive case we abbreviate the invariant case condition

$$(\mathsf{y} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{y})) \wedge \mathsf{y} \not\equiv 0)$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{y} \preceq_{\mathsf{binword}} \mathsf{y}, \mathsf{false}\rangle}\;\text{Identity}}{\left\langle \varphi, \mathsf{pred0}(\mathsf{y}) \preceq_{\mathsf{binword}} \mathsf{y}, \mathsf{false} \vee \Delta^1_{\mathsf{pred0}}(\mathsf{y}) \equiv \mathsf{true}\right\rangle}\;\text{Estimation}$$

To ensure the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword} \\ \quad (\mathsf{y} \equiv \mathsf{succ0}(\mathsf{pred0}(\mathsf{y})) \wedge \mathsf{y} \not\equiv 0) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred0}}(\mathsf{y}) \equiv \mathsf{true}) \end{array}$$

which can be easily proved using the definition of the involved functions. For the second recursive case we abbreviate the invariant case condition

$$(\mathsf{y} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{y})) \wedge \mathsf{y} \not\equiv 1)$$

by φ, and, using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{y} \preceq_{\mathsf{binword}} \mathsf{y}, \mathsf{false} \rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{pred1}(\mathsf{y}) \preceq_{\mathsf{binword}} \mathsf{y}, \mathsf{false} \vee \Delta^1_{\mathsf{pred1}}(\mathsf{y}) \equiv \mathsf{true} \right\rangle}\ \text{Estimation}$$

To ensure the strict relation, we have to prove

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword} \\ \quad (\mathsf{y} \equiv \mathsf{succ1}(\mathsf{pred1}(\mathsf{y})) \wedge \mathsf{y} \not\equiv 1) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{pred1}}(\mathsf{y}) \equiv \mathsf{true}) \end{array}$$

which can be easily proved using the definition of the involved functions.

10.5 $<_{\mathsf{binword}}$: binword × binword → bool

$<_{\mathsf{binword}}$ computes the less-than-relation on binary words and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword} \\ \quad (\mathsf{x} <_{\mathsf{binword}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow (\mathsf{y} - \mathsf{x}) \not\equiv 0 \end{array}$$

Since this is a non-recursive definition, we are done.

10.6 $\leq_{\mathsf{binword}}$: binword × binword → bool

$\leq_{\mathsf{binword}}$ computes the less-than-or-equal-relation on binary words and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword} \\ \quad (\mathsf{x} \leq_{\mathsf{binword}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow ((\mathsf{x} <_{\mathsf{binword}} \mathsf{y}) \equiv \mathsf{true} \vee \mathsf{x} \equiv \mathsf{y}) \end{array}$$

Since this is a non-recursive definition, we are done.

10.7 $>_{\mathsf{binword}}$: binword × binword → bool

$>_{\mathsf{binword}}$ computes the greater-than-relation on binary words and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword} \\ \quad (\mathsf{x} >_{\mathsf{binword}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow (\mathsf{y} <_{\mathsf{binword}} \mathsf{x}) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive definition, we are done.

10.8 $\geq_{\mathsf{binword}}$: $\mathsf{binword} \times \mathsf{binword} \rightarrow \mathsf{bool}$

$\geq_{\mathsf{binword}}$ computes the greater-than-or-equal-relation on binary words and is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}, \mathsf{y} : \mathsf{binword} \\ \quad (\mathsf{x} \geq_{\mathsf{binword}} \mathsf{y}) \equiv \mathsf{true} \leftrightarrow (\mathsf{y} \leq_{\mathsf{binword}} \mathsf{x}) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive definition, we are done.

11

Commutative Trees, tree

This specification of commutative trees (of nats), tree, uses two constructor functions $\mathsf{nil} :\to \mathsf{tree}$, generating the empty tree and $\mathsf{cons} : \mathsf{nat} \times \mathsf{tree} \times \mathsf{tree} \to \mathsf{tree}$, creating a tree from a nat and two existing trees. Equality on tree is specified by the axioms:

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}\!:\!\mathsf{tree}\\ \quad \mathsf{nil} \not\equiv \mathsf{cons}(\mathsf{x}, \mathsf{A}, \mathsf{B})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}, \mathsf{y}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\!:\!\mathsf{tree}\\ \quad \mathsf{cons}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{cons}(\mathsf{y}, \mathsf{C}, \mathsf{D})\\ \quad\quad \leftrightarrow \left(\begin{array}{c} \mathsf{x} \equiv \mathsf{y} \wedge \\ \left(\begin{array}{c} (\mathsf{A} \equiv \mathsf{C} \wedge \mathsf{B} \equiv \mathsf{D}) \vee \\ (\mathsf{A} \equiv \mathsf{D} \wedge \mathsf{B} \equiv \mathsf{C}) \end{array} \right) \end{array} \right)\end{array}$$

By the above specification we have defined a non-freely generated data type. Hence, we must prove the constructor function cons to be size increasing by using the respective implementation specification. Furthermore, the strictness predicates $\Theta^2_{\mathsf{cons}} : \mathsf{nat} \times \mathsf{tree} \times \mathsf{tree} \to \mathsf{bool}$ and $\Theta^3_{\mathsf{cons}} : \mathsf{nat} \times \mathsf{tree} \times \mathsf{tree} \to \mathsf{bool}$, as well as the minimal representation predicate $\Gamma_{\mathsf{cons}} : \mathsf{nat} \times \mathsf{tree} \times \mathsf{tree} \to \mathsf{bool}$ have to be synthesized.

The implementation specification is automatically generated using the constructor functions $\mathsf{nil}_{\mathrm{I}} :\to \mathsf{tree}_{\mathrm{I}}$ $\mathsf{cons}_{\mathrm{I}} : \mathsf{nat} \times \mathsf{tree}_{\mathrm{I}} \times \mathsf{tree}_{\mathrm{I}} \to \mathsf{tree}_{\mathrm{I}}$, and the new equality predicate $\mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}} : \mathsf{tree}_{\mathrm{I}} \times \mathsf{tree}_{\mathrm{I}} \to \mathsf{bool}$.

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \mathsf{nil}_{\mathrm{I}} \not\equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A},\mathsf{B})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B},\mathsf{C},\mathsf{D}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A},\mathsf{B}) \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{y},\mathsf{C},\mathsf{D})\\ \qquad \leftrightarrow (\mathsf{x}\equiv\mathsf{y} \wedge \mathsf{A}\equiv\mathsf{C} \wedge \mathsf{B}\equiv\mathsf{D})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{nil}_{\mathrm{I}},\mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A},\mathsf{B})) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B},\mathsf{C},\mathsf{D}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{A},\mathsf{B}),\mathsf{Cons}_{\mathrm{I}}(\mathsf{y},\mathsf{C},\mathsf{D})) \equiv \mathsf{true}\\ \qquad \leftrightarrow \left(\begin{array}{c}\mathsf{x}\equiv\mathsf{y}\wedge\\ \left(\begin{array}{c}(\mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A},\mathsf{C})\equiv\mathsf{true}\wedge \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B},\mathsf{D})\equiv\mathsf{true})\vee\\ (\mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A},\mathsf{D})\equiv\mathsf{true}\wedge \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B},\mathsf{C})\equiv\mathsf{true})\end{array}\right)\end{array}\right)\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \rightarrow \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B},\mathsf{C}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad (\mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B},\mathsf{C}) \equiv \mathsf{true})\\ \qquad \rightarrow \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A},\mathsf{C}) \equiv \mathsf{true}\end{array}$$

Since $\mathsf{tree}_{\mathrm{I}}$ is freely generated, the strictness predicates $\theta^2_{\mathsf{cons}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{tree}_{\mathrm{I}} \times \mathsf{tree}_{\mathrm{I}} \rightarrow \mathsf{bool}$ and $\theta^3_{\mathsf{cons}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{tree}_{\mathrm{I}} \times \mathsf{tree}_{\mathrm{I}} \rightarrow \mathsf{bool}$, as well as the minimal representation predicate $\gamma_{\mathsf{cons}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{tree}_{\mathrm{I}} \times \mathsf{tree}_{\mathrm{I}} \rightarrow \mathsf{bool}$ are defined by:

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \theta^2_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x},\mathsf{A},\mathsf{B}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \theta^3_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x},\mathsf{A},\mathsf{B}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \gamma_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x},\mathsf{A},\mathsf{B}) \equiv \mathsf{true}\end{array}$$

In addition, all constructor functions of $\mathsf{tree}_{\mathrm{I}}$ are non-overlapping. Hence, for the constructor function $\mathsf{cons}_{\mathrm{I}}$ we introduce three destructor functions, $\mathsf{get\_nat}_{\mathrm{I}} : \mathsf{tree}_{\mathrm{I}} \rightarrow \mathsf{nat}$ for the first argument of $\mathsf{cons}_{\mathrm{I}}$, $\mathsf{left\_tree}_{\mathrm{I}} : \mathsf{tree}_{\mathrm{I}} \rightarrow \mathsf{tree}_{\mathrm{I}}$ for the second argument of $\mathsf{cons}_{\mathrm{I}}$, and $\mathsf{right\_tree}_{\mathrm{I}} : \mathsf{tree}_{\mathrm{I}} \rightarrow \mathsf{tree}_{\mathrm{I}}$ for the third argument of $\mathsf{cons}_{\mathrm{I}}$. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l}\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B},\mathsf{C}\!:\!\mathsf{tree}_{\mathrm{I}}\\ \quad \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{x},\mathsf{B},\mathsf{C}) \rightarrow \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}),\mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}),\mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))\end{array}$$

$$\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{nil}_{\mathrm{I}}) \equiv 0\ (\equiv \triangledown_{\mathsf{nat}})$$

$$\mathsf{left\_tree}_{\mathrm{I}}(\mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{nil}_{\mathrm{I}}$$

$$\mathsf{right\_tree}_{\mathrm{I}}(\mathsf{nil}_{\mathrm{I}}) \equiv \mathsf{nil}_{\mathrm{I}}$$

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B}, \mathsf{C}\!:\!\mathsf{tree}_{\mathrm{I}}$$
$$\mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{x}, \mathsf{B}, \mathsf{C}) \to \gamma_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A})) \equiv \mathsf{true}$$

Now, $\mathsf{left\_tree}_{\mathrm{I}}$ and $\mathsf{right\_tree}_{\mathrm{I}}$ are both 1-bounded with difference predicates $\Delta^{\mathrm{I2}}_{\mathsf{left\_tree}_{\mathrm{I}}} : \mathsf{tree}_{\mathrm{I}} \to \mathsf{bool}$ and $\Delta^{\mathrm{I3}}_{\mathsf{right\_tree}_{\mathrm{I}}} : \mathsf{tree}_{\mathrm{I}} \to \mathsf{bool}$, defined by

$$\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}$$
$$\Delta^{\mathrm{I1}}_{\mathsf{left\_tree}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))$$

$$\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}$$
$$\Delta^{\mathrm{I1}}_{\mathsf{right\_tree}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))$$

Furthermore, the function $\mathsf{term\_size}_{\mathsf{tree}_{\mathrm{I}}} : \mathsf{tree}_{\mathrm{I}} \to \mathsf{nat}$ is synthesized by:

$$\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}$$
$$\mathsf{A} \equiv \mathsf{nil}_{\mathrm{I}} \to \mathsf{term\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) \equiv 0$$

$$\begin{aligned}&\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to \mathsf{term\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) \equiv\\ &\qquad\qquad \mathsf{succ}(\mathsf{term\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A})) + \mathsf{term\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A})))\end{aligned}$$

In order to have easier proofs, we specify a function $\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}} : \mathsf{tree}_{\mathrm{I}} \to \mathsf{nat}$, by

$$\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}$$
$$\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{pred}(\mathsf{length}(\mathsf{A})),$$

where $\mathsf{length} : \mathsf{tree}_{\mathrm{I}} \to \mathsf{nat}$ is defined constructively by

$$\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}$$
$$\mathsf{A} \equiv \mathsf{nil}_{\mathrm{I}} \to \mathsf{length}(\mathsf{A}) \equiv 0$$

$$\begin{aligned}&\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to \mathsf{length}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{length}(\mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A})) + \mathsf{length}(\mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A})))\end{aligned}$$

The specification of length is case-distinct, as proved by

$$\forall\, \mathsf{A}\!:\!\mathsf{tree}_{\mathrm{I}}$$
$$\neg \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{nil}_{\mathrm{I}} \wedge \\ \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A})) \end{array} \right)$$

Furthermore, the recursion ordering of length is well-founded. To prove that we use the Estimation Calculus. There is only one recursive case with two recursive calls of length. Now, we abbreviate the case condition

$$\mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Then, for the first recursive call the derivation in the Estimation Calculus is given by

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{tree}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{left\_tree}_{\mathrm{I}}}(\mathsf{A})\right\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{aligned}&\forall\, \mathsf{A}:\mathsf{tree}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to (\mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{left\_tree}_{\mathrm{I}}}(\mathsf{A}))\end{aligned}$$

which can be simplified to

$$\begin{aligned}&\forall\, \mathsf{A}:\mathsf{tree}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A})).\end{aligned}$$

Similarly, for the second recursive call we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}_{\mathrm{I}}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{tree}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{right\_tree}_{\mathrm{I}}}(\mathsf{A})\right\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{aligned}&\forall\, \mathsf{A}:\mathsf{tree}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to (\mathsf{false} \vee \Delta^{\mathrm{I1}}_{\mathsf{right\_tree}_{\mathrm{I}}}(\mathsf{A}))\end{aligned}$$

which can be simplified to

$$\begin{aligned}&\forall\, \mathsf{A}:\mathsf{tree}_{\mathrm{I}}\\ &\quad \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A}))\\ &\qquad \to \mathsf{A} \equiv \mathsf{cons}_{\mathrm{I}}(\mathsf{get\_nat}_{\mathrm{I}}(\mathsf{A}), \mathsf{left\_tree}_{\mathrm{I}}(\mathsf{A}), \mathsf{right\_tree}_{\mathrm{I}}(\mathsf{A})).\end{aligned}$$

Now, we need to prove that the above axiomatization of $\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}$ computes the minimal size of a tree, indeed. Therefore we need to show the following proof obligations

$$\begin{aligned}&\forall\, \mathsf{A}, \mathsf{B}:\mathsf{tree}_{\mathrm{I}}\\ &\quad \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \to (\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B})) \equiv \mathsf{true}\end{aligned}$$

$$\begin{aligned}&\forall\, \mathsf{A}:\mathsf{tree}_{\mathrm{I}}\ \exists\, \mathsf{B}:\mathsf{tree}_{\mathrm{I}}\\ &\quad \mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \wedge (\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) \geq_{\mathsf{nat}} \mathsf{term\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B})) \equiv \mathsf{true}\end{aligned}$$

$$\forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}_{\mathrm{I}}$$
$$\mathsf{Eq}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true} \to \mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) \equiv \mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B})$$

Next, we need to show that cons denotes a size increasing constructor function. To do that, we prove:

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}_{\mathrm{I}}$$
$$(\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{cons}_{\mathrm{I}}(\mathsf{x}, \mathsf{A}, \mathsf{B}))) \equiv \mathsf{true} \wedge$$
$$(\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B}) \leq_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{cons}_{\mathrm{I}}(\mathsf{x}, \mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}$$

Finally, we need to define the strictness predicates $\Theta^2_{\mathsf{cons}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{tree}_{\mathrm{I}} \times \mathsf{tree}_{\mathrm{I}} \to \mathsf{bool}$ and $\Theta^3_{\mathsf{cons}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{tree}_{\mathrm{I}} \times \mathsf{tree}_{\mathrm{I}} \to \mathsf{bool}$, as well as the minimal representation predicate $\Gamma_{\mathsf{cons}_{\mathrm{I}}} : \mathsf{nat} \times \mathsf{tree}_{\mathrm{I}} \times \mathsf{tree}_{\mathrm{I}} \to \mathsf{bool}$. We suggest the following definitions:

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}_{\mathrm{I}}$$
$$\Theta^2_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}_{\mathrm{I}}$$
$$\Theta^3_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}_{\mathrm{I}}$$
$$\Gamma_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

However, we have to prove that our suggestions really define the strictness predicates and the minimal representation predicate. Hence, we need to show that

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}_{\mathrm{I}}$$
$$\Theta^2_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$
$$\leftrightarrow (\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{cons}_{\mathrm{I}}(\mathsf{x}, \mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}_{\mathrm{I}}$$
$$\Theta^3_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$
$$\leftrightarrow (\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B}) <_{\mathsf{nat}} \mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{cons}_{\mathrm{I}}(\mathsf{x}, \mathsf{A}, \mathsf{B}))) \equiv \mathsf{true}$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}_{\mathrm{I}}$$
$$\Gamma_{\mathsf{cons}_{\mathrm{I}}}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$
$$\leftrightarrow \mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{cons}_{\mathrm{I}}(\mathsf{x}, \mathsf{A}, \mathsf{B})) \equiv$$
$$\mathsf{succ}(\mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{A}) + \mathsf{min\_size}_{\mathsf{tree}_{\mathrm{I}}}(\mathsf{B}))$$

Having done so, we know for our original specification tree that the constructor function cons is size increasing, and we can translate the strictness predicates and the minimal representation predicate into the original specification. Hence, we obtain:

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}$$
$$\Theta^2_{\mathsf{cons}}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}$$
$$\Theta^3_{\mathsf{cons}}(\mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A}, \mathsf{B} : \mathsf{tree}\ \Gamma_{\mathsf{cons}}(\mathsf{x}, \mathsf{A}, \mathsf{B}) \equiv \mathsf{true}$$

The data type tree possesses non-overlapping constructor functions. Hence, we can use the simplified construction scheme for the destructor functions. For the constructor function cons we introduce three destructor functions $\mathsf{get\_nat} : \mathsf{tree} \to \mathsf{nat}$ for the first argument of cons, $\mathsf{left\_tree} : \mathsf{tree} \to \mathsf{tree}$ for the second argument of cons, and $\mathsf{right\_tree} : \mathsf{tree} \to \mathsf{tree}$ for the third argument of cons. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l}\forall\, x{:}\mathsf{nat}\ \forall\, A, B, C{:}\mathsf{tree}\\ \quad A \equiv \mathsf{cons}(x, B, C) \to A \equiv \mathsf{cons}(\mathsf{get\_nat}(A), \mathsf{left\_tree}(A), \mathsf{right\_tree}(A)),\end{array}$$

$$\mathsf{get\_nat}(\mathsf{nil}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\mathsf{left\_tree}(\mathsf{nil}) \equiv \mathsf{nil}$$

$$\mathsf{right\_tree}(\mathsf{nil}) \equiv \mathsf{nil}$$

$$\begin{array}{l}\forall\, x{:}\mathsf{nat}\ \forall\, A, B, C{:}\mathsf{tree}\\ \quad A \equiv \mathsf{cons}(x, B, C)\\ \qquad \to \Gamma_{\mathsf{cons}}(\mathsf{get\_nat}(A), \mathsf{left\_tree}(A), \mathsf{right\_tree}(A)) \equiv \mathsf{true},\end{array}$$

Both reflexive destructor functions of the constructor function cons, left_tree and right_tree, are 1-bounded, and their difference predicates $\Delta^1_{\mathsf{left\_tree}} : \mathsf{tree} \to \mathsf{bool}$ and $\Delta^1_{\mathsf{right\_tree}} : \mathsf{tree} \to \mathsf{bool}$, are defined by

$$\begin{array}{l}\forall\, A{:}\mathsf{tree}\\ \quad \Delta^1_{\mathsf{left\_tree}}(A) \equiv \mathsf{true}\\ \qquad \leftrightarrow A \equiv \mathsf{cons}(\mathsf{get\_nat}(A), \mathsf{left\_tree}(A), \mathsf{right\_tree}(A))\end{array}$$

$$\begin{array}{l}\forall\, A{:}\mathsf{tree}\\ \quad \Delta^1_{\mathsf{right\_tree}}(A) \equiv \mathsf{true}\\ \qquad \leftrightarrow A \equiv \mathsf{cons}(\mathsf{get\_nat}(A), \mathsf{left\_tree}(A), \mathsf{right\_tree}(A))\end{array}$$

For the data type tree we will give constructive function and predicate specifications for count, height, leafcount, delete, $<_{\mathsf{tree}}$, $\leq_{\mathsf{tree}}$, $>_{\mathsf{tree}}$, and $\geq_{\mathsf{tree}}$.

11.1 count : tree → nat

count computes the number of nodes in a tree, and it is defined by:

$$\begin{array}{l}\forall\, A{:}\mathsf{tree}\\ \quad A \equiv \mathsf{nil} \to \mathsf{count}(A) \equiv 0\end{array}$$

$$\begin{array}{l}\forall\, A{:}\mathsf{tree}\\ \quad A \equiv \mathsf{cons}(\mathsf{get\_nat}(A), \mathsf{left\_tree}(A), \mathsf{right\_tree}(A))\\ \qquad \to \mathsf{count}(A) \equiv \mathsf{succ}(\mathsf{count}(\mathsf{left\_tree}(A) + \mathsf{count}(\mathsf{right\_tree}(A)))\end{array}$$

The recursion ordering of count is well-founded. There is only a single recursive definition case with two recursive calls of count. Hence, we abbreviate the invariant case condition

$$A \equiv \mathsf{cons}(\mathsf{get\_nat}(A), \mathsf{left\_tree}(A), \mathsf{right\_tree}(A))$$

by φ, and, using the Estimation Calculus for the first recursive call, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{left\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \\ \qquad \to (\mathsf{false} \lor \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \\ \qquad \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \end{array}$$

For the second recursive call, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \\ \qquad \to (\mathsf{false} \lor \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \\ \qquad \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \end{array}$$

11.2 height : tree → nat

height computes the height of a tree, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{height}(\mathsf{A}) \equiv 0 \end{array}$$

$$\forall\, \mathsf{A}:\mathsf{tree}\; \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \wedge \\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix} \to \mathsf{height}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{height}(\mathsf{right\_tree}(\mathsf{A})))$$

$$\forall\, \mathsf{A}:\mathsf{tree}\; \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \wedge \\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix} \to \mathsf{height}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})))$$

The recursion ordering of height is well-founded. There are two recursive definition cases with three recursive calls in each, however, two are identical. For the first recursive case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \wedge \\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix}$$

by φ, and, using the Estimation Calculus for the first recursive call, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \rangle}\,\mathsf{Identity}}{\langle \varphi, \mathsf{left\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\,\mathsf{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\forall\, \mathsf{A}:\mathsf{tree}\; \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \wedge \\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix} \to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true})$$

which can be simplified to

$$\forall\, \mathsf{A}:\mathsf{tree}\; \begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \wedge \\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix} \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))$$

For the second recursive call, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \rangle}\,\mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true} \right\rangle}\,\mathsf{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right)\\ \qquad\to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right)\\ \qquad\to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\end{array}$$

For the second recursive definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)$$

by φ, and, using the Estimation Calculus for the first recursive call, we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\;\mathsf{Identity}}{\langle \varphi, \mathsf{left\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\;\mathsf{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\\ \qquad\to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\\ \qquad\to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\end{array}$$

For the second recursive call, we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\;\mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\;\mathsf{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\\ \qquad\to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ (\mathsf{height}(\mathsf{left\_tree}(\mathsf{A})) <_{\mathsf{nat}} \mathsf{height}(\mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\\ \qquad\to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\end{array}$$

11.3 leafcount : tree → nat

leafcount computes the number of leaves in a tree, and it is defined by:

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{tree}\\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{leafcount}(\mathsf{A}) \equiv \mathsf{succ}(0)\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{tree}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\\ \qquad\to \mathsf{leafcount}(\mathsf{A}) \equiv (\mathsf{leafcount}(\mathsf{left\_tree}(\mathsf{A}) + \mathsf{leafcount}(\mathsf{right\_tree}(\mathsf{A})))\end{array}$$

The recursion ordering of leafcount is well-founded. There is only a single recursive definition case with two recursive calls of leafcount. Hence, we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))$$

by φ, and, using the Estimation Calculus for the first recursive call, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{left\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{tree}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\\ \qquad\to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{tree}\\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\\ \qquad\to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\end{array}$$

For the second recursive call, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l} \forall\, \mathsf{A}{:}\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \\ \qquad \to (\mathsf{false} \lor \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{A}{:}\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \\ \qquad \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \end{array}$$

11.4 $\mathsf{delete} : \mathsf{nat} \times \mathsf{tree} \to \mathsf{tree}$

delete deletes all subtrees with the specified object as node. It is defined by:

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{nil} \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{nil} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{tree} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \land \\ \mathsf{get\_nat}(\mathsf{A}) \equiv \mathsf{x} \end{array} \right) \\ \qquad \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{nil} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{tree} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \land \\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x} \end{array} \right) \\ \qquad \to \mathsf{delete}(\mathsf{x}, \mathsf{A}) \equiv \\ \qquad\qquad \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A}))) \end{array}$$

The recursion ordering of delete is well-founded. There is only a single recursive definition case with two recursive calls of delete. Hence, we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \land \\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x} \end{array} \right)$$

by φ, and, using the Estimation Calculus for the first recursive call, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{left\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \lor \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x}\end{array}\right)\\ \quad\quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x}\end{array}\right)\\ \quad\quad \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\end{array}$$

For the second recursive call, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{right\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \mathsf{Estimation}$$

In order to ensure the strict relation, we need to prove

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x}\end{array}\right)\\ \quad\quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{tree}\\ \quad\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x}\end{array}\right)\\ \quad\quad \to \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\end{array}$$

In addition, delete denotes a 2-bounded function symbol. First of all, delete is completely specified, as proved by

$$\begin{array}{l}\forall\, \mathsf{x}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{tree}\\ \quad\begin{array}{c}\mathsf{A} \equiv \mathsf{nil}\vee\\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ \mathsf{get\_nat}(\mathsf{A}) \equiv \mathsf{x}\end{array}\right)\vee\\ \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A}))\wedge\\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x}\end{array}\right)\end{array}\end{array}$$

Next, we examine each definition case separately. For the first definition case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{nil}$$

by φ, and we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{nil} \preceq_{\mathsf{tree}} \mathsf{nil}, \mathsf{false} \rangle}\ \text{Identity}}{\langle \varphi, \mathsf{nil} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \rangle}\ \text{Equation 1}$$

For the second definition case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}(\mathsf{A}) \equiv \mathsf{x} \end{pmatrix}$$

by φ, and we obtain the derivation:

$$\cfrac{\cfrac{-}{\langle \varphi, \mathsf{nil} \preceq_{\mathsf{tree}} \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})), \mathsf{true} \rangle}\ \text{Strong Estimation}}{\langle \varphi, \mathsf{nil} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{true} \rangle}\ \text{Equation 1}$$

For the third definition case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \wedge \\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x} \end{pmatrix}$$

by φ. Since this is a recursive definition case, we may assume the additional inference rules

$$\xi_1 \Rightarrow \quad \cfrac{\langle \varphi, \mathsf{left\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \Delta_1 \rangle}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \preceq_{\mathsf{tree}} \mathsf{left\_tree}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \equiv \mathsf{true} \rangle}\ \text{Induction Hypothesis}$$

where ξ_1 is an abbreviation for the formula

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{tree}\ \varphi \to \Delta_1$$

and

$$\xi_2 \Rightarrow \quad \cfrac{\langle \varphi, \mathsf{right\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \Delta_2 \rangle}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \preceq_{\mathsf{tree}} \mathsf{left\_tree}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \equiv \mathsf{true} \rangle}\ \text{Induction Hypothesis}$$

where ξ_2 is an abbreviation for the formula

$$\forall\, \mathsf{x} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{tree}\ \varphi \to \Delta_2$$

as induction hypotheses. Thus, we obtain the derivation:

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{left\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \text{Estimation}}{\langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \preceq_{\mathsf{tree}} \mathsf{left\_tree}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \text{Induction Hypothesis}$$

where to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{tree}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{left\_tree}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved. On the other hand we can derive:

$$\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false}\rangle}\ \text{Identity}}{\left\langle \varphi, \mathsf{right\_tree}(\mathsf{A}) \preceq_{\mathsf{tree}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true}\right\rangle}\ \text{Estimation}}{\left\langle \begin{array}{c} \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \preceq_{\mathsf{tree}} \mathsf{right\_tree}(\mathsf{A}), \\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Induction Hypothesis}$$

where to allow the application of the induction hypothesis, the formula

$$\forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{tree}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{right\_tree}}(\mathsf{A}) \equiv \mathsf{true})$$

needs to be shown. Having derived the above estimation formulas we can now derive:

$$\dfrac{\dfrac{\begin{array}{c} \langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \preceq_{\mathsf{tree}} \mathsf{left\_tree}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \equiv \mathsf{true}\rangle\,, \\ \langle \varphi, \mathsf{delete}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \preceq_{\mathsf{tree}} \mathsf{right\_tree}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \equiv \mathsf{true}\rangle \end{array}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A}))) \\ \preceq_{\mathsf{tree}} \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})), \\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{cons}}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A}))) \preceq_{\mathsf{tree}} \mathsf{A}, \\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{cons}}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{delete}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})), \mathsf{delete}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where in order to apply the Weak Embedding Rule, the formula

$$\forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{tree}\ \varphi \to \Gamma_{\mathsf{cons}}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \equiv \mathsf{true}$$

has to be proved.

Now, we can synthesize the difference predicate $\Delta^2_{\mathsf{delete}} : \mathsf{nat} \times \mathsf{tree} \to \mathsf{bool}$, using the same case analysis as the specification of delete, and using the simplified difference formulas from each derivation in the Estimation Calculus:

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{tree} \\ \quad \mathsf{A} \equiv \mathsf{nil} \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{tree} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \land \\ \mathsf{get\_nat}(\mathsf{A}) \equiv \mathsf{x} \end{array} \right) \\ \quad\quad \to \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{x}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{tree} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{cons}(\mathsf{get\_nat}(\mathsf{A}), \mathsf{left\_tree}(\mathsf{A}), \mathsf{right\_tree}(\mathsf{A})) \land \\ \mathsf{get\_nat}(\mathsf{A}) \not\equiv \mathsf{x} \end{array} \right) \\ \quad\quad \to \left(\begin{array}{l} \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{A}) \equiv \mathsf{true} \\ \quad \leftrightarrow \left(\begin{array}{l} \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{left\_tree}(\mathsf{A})) \equiv \mathsf{true} \lor \\ \Delta^2_{\mathsf{delete}}(\mathsf{x}, \mathsf{right\_tree}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right) \end{array} \right) \end{array}$$

11.5 $<_{\mathsf{tree}}$: tree × tree → bool

$<_{\mathsf{tree}}$ computes the less-than-relation on trees and is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{tree} \\ \quad (\mathsf{A} <_{\mathsf{tree}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{count}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{count}(\mathsf{B})) \equiv \mathsf{true} \end{array}$$

This predicate denotes a well-founded relation. Since this is a non-recursive specification, we are done.

11.6 $\leq_{\mathsf{tree}}$: tree × tree → bool

$\leq_{\mathsf{tree}}$ computes the less-than-or-equal-relation on trees and is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{tree} \\ \quad (\mathsf{A} \leq_{\mathsf{tree}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{count}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{count}(\mathsf{B})) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive specification, we are done.

11.7 $>_{\mathsf{tree}}$: tree × tree → bool

$>_{\mathsf{tree}}$ computes the greater-than-relation on trees and is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}, \mathsf{B}{:}\mathsf{tree} \\ \quad (\mathsf{A} >_{\mathsf{tree}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} <_{\mathsf{tree}} \mathsf{A}) \equiv \mathsf{true} \end{array}$$

Since this is a non-recursive specification, we are done.

11.8 $\geq_{\mathsf{tree}}$: $\mathsf{tree} \times \mathsf{tree} \to \mathsf{bool}$

$\geq_{\mathsf{tree}}$ computes the greater-than-or-equal-relation on trees and is defined by:

$$\begin{array}{l}\forall\ \mathsf{A}, \mathsf{B} : \mathsf{tree} \\ \quad (\mathsf{A} \geq_{\mathsf{tree}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} \leq_{\mathsf{tree}} \mathsf{A}) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive specification, we are done.

12

Arrays, array

This specification of arrays with nats as index as well as entry data type, array, uses two constructor functions $\mathsf{void} :\to \mathsf{array}$, generating the empty (initial) array, and $\mathsf{put} : \mathsf{nat} \times \mathsf{nat} \times \mathsf{array} \to \mathsf{array}$, for the update operation of an array. Equality on array is specified using an auxiliary predicate $\in: \mathsf{nat} \times \mathsf{array} \to \mathsf{bool}$, using an auxiliary function $\mathsf{aref} : \mathsf{nat} \times \mathsf{array} \to \mathsf{nat}$, and by the axioms:

$$\forall\, \mathsf{i}\!:\!\mathsf{nat}\quad \mathsf{i} \notin \mathsf{void}$$

$$\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\quad \mathsf{i} \in \mathsf{put}(\mathsf{j},\mathsf{x},\mathsf{A}) \leftrightarrow (\mathsf{i} \equiv \mathsf{j} \vee \mathsf{i} \in \mathsf{A})$$

$$\forall\, \mathsf{i}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\quad \mathsf{aref}(\mathsf{i},\mathsf{put}(\mathsf{i},\mathsf{x},\mathsf{A})) \equiv \mathsf{x}$$

$$\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\quad \mathsf{i} \not\equiv \mathsf{j} \to \mathsf{aref}(\mathsf{i},\mathsf{put}(\mathsf{j},\mathsf{x},\mathsf{A})) \equiv \mathsf{aref}(\mathsf{i},\mathsf{A})$$

$$\forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{array}\quad \mathsf{A} \equiv \mathsf{B} \leftrightarrow (\forall\, \mathsf{i}\!:\!\mathsf{nat}\ (\mathsf{i} \in \mathsf{A} \vee \mathsf{i} \in \mathsf{B}) \to (\mathsf{i} \in \mathsf{A} \wedge \mathsf{i} \in \mathsf{B} \wedge \mathsf{aref}(\mathsf{i},\mathsf{A}) \equiv \mathsf{aref}(\mathsf{i},\mathsf{B})))$$

By the above specification we have defined a non-freely generated data type. Hence, we must prove the constructor function put to be size increasing by using the respective implementation specification. Furthermore, the strictness predicate $\Theta^3_{\mathsf{put}} : \mathsf{nat} \times \mathsf{nat} \times \mathsf{array} \to \mathsf{bool}$ and the minimal representation predicate $\Gamma_{\mathsf{put}} : \mathsf{nat} \times \mathsf{nat} \times \mathsf{array} \to \mathsf{bool}$ have to be synthesized.

The implementation specification is automatically generated using the constructor functions $\mathsf{void}_\mathrm{I} :\to \mathsf{array}_\mathrm{I}$, $\mathsf{put}_\mathrm{I} : \mathsf{nat} \times \mathsf{nat} \times \mathsf{array}_\mathrm{I} \to \mathsf{array}_\mathrm{I}$, as well as the new equality predicate $\mathsf{Eq}_{\mathsf{array}_\mathrm{I}} : \mathsf{array}_\mathrm{I} \times \mathsf{array}_\mathrm{I} \to \mathsf{bool}$.

$$\begin{array}{l}\forall\, \mathsf{i}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \mathsf{void}_\mathrm{I} \not\equiv \mathsf{put}_\mathrm{I}(\mathsf{i},\mathsf{x},\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{x},\mathsf{y}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \mathsf{put}_\mathrm{I}(\mathsf{i},\mathsf{x},\mathsf{A}) \equiv \mathsf{put}_\mathrm{I}(\mathsf{j},\mathsf{y},\mathsf{B}) \to (\mathsf{i} \equiv \mathsf{j} \wedge \mathsf{x} \equiv \mathsf{y} \wedge \mathsf{A} \equiv \mathsf{B})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i}\!:\!\mathsf{nat}\\ \quad \mathsf{i} \notin_\mathrm{I} \mathsf{void}_\mathrm{I}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \mathsf{i} \in_\mathrm{I} \mathsf{put}_\mathrm{I}(\mathsf{j},\mathsf{x},\mathsf{A}) \leftrightarrow (\mathsf{i} \equiv \mathsf{j} \vee \mathsf{i} \in_\mathrm{I} \mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \mathsf{aref}_\mathrm{I}(\mathsf{i},\mathsf{put}_\mathrm{I}(\mathsf{i},\mathsf{x},\mathsf{A})) \equiv \mathsf{x}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \mathsf{i} \not\equiv \mathsf{j} \to \mathsf{aref}_\mathrm{I}(\mathsf{i},\mathsf{put}_\mathrm{I}(\mathsf{j},\mathsf{x},\mathsf{A})) \equiv \mathsf{aref}_\mathrm{I}(\mathsf{i},\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true}\\ \qquad \leftrightarrow (\forall\, \mathsf{i}\!:\!\mathsf{nat}\ (\mathsf{i} \in_\mathrm{I} \mathsf{A} \vee \mathsf{i} \in_\mathrm{I} \mathsf{B}) \to (\mathsf{i} \in_\mathrm{I} \mathsf{A} \wedge \mathsf{i} \in_\mathrm{I} \mathsf{B} \wedge \mathsf{aref}_\mathrm{I}(\mathsf{i},\mathsf{A}) \equiv \mathsf{aref}_\mathrm{I}(\mathsf{i},\mathsf{B})))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{A},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \to \mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{B},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A},\mathsf{B},\mathsf{C}\!:\!\mathsf{array}_\mathrm{I}\\ \quad (\mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true} \wedge \mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{B},\mathsf{C}) \equiv \mathsf{true})\\ \qquad \to \mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{A},\mathsf{C}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{array}_\mathrm{I}\\ \quad (\mathsf{i} \equiv \mathsf{j} \wedge \mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true})\\ \qquad \to (\mathsf{i} \in_\mathrm{I} \mathsf{A} \leftrightarrow \mathsf{j} \in_\mathrm{I} \mathsf{B})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{array}_\mathrm{I}\\ \quad (\mathsf{i} \equiv \mathsf{j} \wedge \mathsf{Eq}_{\mathsf{array}_\mathrm{I}}(\mathsf{A},\mathsf{B}) \equiv \mathsf{true})\\ \qquad \to \mathsf{aref}_\mathrm{I}(\mathsf{i},\mathsf{A}) \equiv \mathsf{aref}_\mathrm{I}(\mathsf{j},\mathsf{B})\end{array}$$

Since $\mathsf{array}_\mathrm{I}$ is freely generated, the strictness predicate $\theta^3_{\mathsf{put}_\mathrm{I}} : \mathsf{nat} \times \mathsf{nat} \times \mathsf{array}_\mathrm{I} \to \mathsf{bool}$ as well as the minimal representation predicate $\gamma_{\mathsf{put}_\mathrm{I}} : \mathsf{nat} \times \mathsf{nat} \times \mathsf{array}_\mathrm{I} \to \mathsf{bool}$ are defined by:

$$\begin{array}{l}\forall\, \mathsf{i}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \theta^3_{\mathsf{put}_\mathrm{I}}(\mathsf{i},\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}_\mathrm{I}\\ \quad \gamma_{\mathsf{put}_\mathrm{I}}(\mathsf{i},\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\end{array}$$

In addition, the constructor functions of $\mathsf{array_I}$ are non-overlapping. Hence, for the constructor function $\mathsf{put_I}$ we introduce two destructor functions $\mathsf{index_I} : \mathsf{array_I} \to \mathsf{nat}$ for the first argument of $\mathsf{put_I}$, $\mathsf{entry_I} : \mathsf{array_I} \to \mathsf{nat}$ for the second argument of $\mathsf{put_I}$ and $\mathsf{sub_I} : \mathsf{array_I} \to \mathsf{array_I}$ for the third argument of $\mathsf{put_I}$. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l} \forall\, \mathsf{i}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{array_I} \\ \quad \mathsf{A} \equiv \mathsf{put_I}(\mathsf{i},\mathsf{x},\mathsf{B}) \to \mathsf{A} \equiv \mathsf{put_I}(\mathsf{index_I}(\mathsf{A}),\mathsf{entry_I}(\mathsf{A}),\mathsf{sub_I}(\mathsf{A})) \end{array}$$

$$\mathsf{index_I}(\mathsf{void_I}) \equiv \mathsf{0}\ (\equiv \triangledown_{\mathsf{nat}})$$

$$\mathsf{entry_I}(\mathsf{void_I}) \equiv \mathsf{0}\ (\equiv \triangledown_{\mathsf{nat}})$$

$$\mathsf{sub_I}(\mathsf{void_I}) \equiv \mathsf{void_I}$$

$$\begin{array}{l} \forall\, \mathsf{i}\!:\!\mathsf{nat}\ \forall\, \mathsf{x}\!:\!\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}\!:\!\mathsf{array_I} \\ \quad \mathsf{A} \equiv \mathsf{put_I}(\mathsf{i},\mathsf{x},\mathsf{B}) \to \gamma_{\mathsf{put_I}}(\mathsf{index_I}(\mathsf{A}),\mathsf{entry_I}(\mathsf{A}),\mathsf{sub_I}(\mathsf{A})) \equiv \mathsf{true} \end{array}$$

Now, $\mathsf{sub_I}$ is 1-bounded with difference predicate $\Delta^{I1}_{\mathsf{sub_I}} : \mathsf{array_I} \to \mathsf{bool}$, defined by

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{array_I} \\ \quad \Delta^{I1}_{\mathsf{sub_I}}(\mathsf{A}) \equiv \mathsf{true} \leftrightarrow \mathsf{A} \equiv \mathsf{put_I}(\mathsf{index_I}(\mathsf{A}),\mathsf{entry_I}(\mathsf{A}),\mathsf{sub_I}(\mathsf{A})) \end{array}$$

Furthermore, the function $\mathsf{term\_size_{array_I}} : \mathsf{array_I} \to \mathsf{nat}$ is synthesized by:

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{array_I} \\ \quad \mathsf{A} \equiv \mathsf{void_I} \to \mathsf{term\_size_{array_I}}(\mathsf{A}) \equiv \mathsf{0} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{array_I} \\ \quad \mathsf{A} \equiv \mathsf{put_I}(\mathsf{index_I}(\mathsf{A}),\mathsf{entry_I}(\mathsf{A}),\mathsf{sub_I}(\mathsf{A})) \\ \qquad \to \mathsf{term\_size_{array_I}}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{term\_size_{array_I}}(\mathsf{sub_I}(\mathsf{A}))) \end{array}$$

In order to have easier proofs, we specify a function $\mathsf{min\_size_{array_I}} : \mathsf{array_I} \to \mathsf{nat}$, by

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{array_I} \\ \quad \mathsf{A} \equiv \mathsf{void_I} \to \mathsf{min\_size_{array_I}}(\mathsf{A}) \equiv \mathsf{0} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{array_I} \\ \quad (\mathsf{A} \equiv \mathsf{put_I}(\mathsf{index_I}(\mathsf{A}),\mathsf{entry_I}(\mathsf{A}),\mathsf{sub_I}(\mathsf{A})) \wedge \mathsf{index_I}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{sub_I}(\mathsf{A})) \\ \qquad \to \mathsf{min\_size_{array_I}}(\mathsf{A}) \equiv \mathsf{min\_size_{array_I}}(\mathsf{sub_I}(\mathsf{A})) \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{array_I} \\ \quad (\mathsf{A} \equiv \mathsf{put_I}(\mathsf{index_I}(\mathsf{A}),\mathsf{entry_I}(\mathsf{A}),\mathsf{sub_I}(\mathsf{A})) \wedge \mathsf{index_I}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{sub_I}(\mathsf{A})) \\ \qquad \to \mathsf{min\_size_{array_I}}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{min\_size_{array_I}}(\mathsf{sub_I}(\mathsf{A}))) \end{array}$$

The specification of $\mathsf{min\_size_{array_I}}$ is case-distinct, as proved by

$$\begin{array}{l} \forall\, \mathsf{A}\!:\!\mathsf{array_I} \\ \quad \neg \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{void_I} \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put_I}(\mathsf{index_I}(\mathsf{A}),\mathsf{entry_I}(\mathsf{A}),\mathsf{sub_I}(\mathsf{A})) \wedge \\ \mathsf{index_I}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{sub_I}(\mathsf{A}) \end{array} \right) \end{array} \right) \end{array}$$

$$\forall\, \mathsf{A}\colon \mathsf{array}_{\mathrm{I}} \quad \neg \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{void}_{\mathrm{I}} \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}), \mathsf{entry}_{\mathrm{I}}(\mathsf{A}), \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \mathsf{index}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{sub}_{\mathrm{I}}(\mathsf{A}) \end{array} \right) \end{array} \right)$$

$$\forall\, \mathsf{A}\colon \mathsf{array}_{\mathrm{I}} \quad \neg \left(\begin{array}{c} \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}), \mathsf{entry}_{\mathrm{I}}(\mathsf{A}), \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \mathsf{index}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{sub}_{\mathrm{I}}(\mathsf{A}) \end{array} \right) \wedge \\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}), \mathsf{entry}_{\mathrm{I}}(\mathsf{A}), \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \wedge \\ \mathsf{index}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{sub}_{\mathrm{I}}(\mathsf{A}) \end{array} \right) \end{array} \right)$$

Furthermore, the recursion ordering of $\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}$ is well-founded. To prove that we use the Estimation Calculus. There are two recursive cases with a single recursive call of $\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}$ in each. For the first recursive case we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}), \mathsf{entry}_{\mathrm{I}}(\mathsf{A}), \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{index}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{sub}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Then, using the Estimation Calculus, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{sub}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{array}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{sub}_{\mathrm{I}}}(\mathsf{A}) \right\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l} \forall\, \mathsf{A}\colon \mathsf{array}_{\mathrm{I}} \\ \quad (\mathsf{A} \equiv \mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}), \mathsf{entry}_{\mathrm{I}}(\mathsf{A}), \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{index}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \\ \quad\quad \rightarrow (\mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{sub}_{\mathrm{I}}}(\mathsf{A})) \end{array}$$

which can be simplified to

$$\begin{array}{l} \forall\, \mathsf{A}\colon \mathsf{array}_{\mathrm{I}} \\ \quad (\mathsf{A} \equiv \mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}), \mathsf{entry}_{\mathrm{I}}(\mathsf{A}), \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{index}_{\mathrm{I}}(\mathsf{A}) \in_{\mathrm{I}} \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \\ \quad\quad \rightarrow \mathsf{A} \equiv \mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}), \mathsf{entry}_{\mathrm{I}}(\mathsf{A}), \mathsf{sub}_{\mathrm{I}}(\mathsf{A})). \end{array}$$

Similarly, for the second recursive case, we abbreviate the case condition

$$(\mathsf{A} \equiv \mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}), \mathsf{entry}_{\mathrm{I}}(\mathsf{A}), \mathsf{sub}_{\mathrm{I}}(\mathsf{A})) \wedge \mathsf{index}_{\mathrm{I}}(\mathsf{A}) \notin_{\mathrm{I}} \mathsf{sub}_{\mathrm{I}}(\mathsf{A}))$$

by φ. Then, using the Estimation Calculus, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\left\langle \varphi, \mathsf{sub}_{\mathrm{I}}(\mathsf{A}) \preceq_{\mathsf{array}_{\mathrm{I}}} \mathsf{A}, \mathsf{false} \vee \Delta^{\mathrm{I}1}_{\mathsf{sub}_{\mathrm{I}}}(\mathsf{A}) \right\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{array}_{\mathrm{I}}\\ \quad(\mathsf{A}\equiv\mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}),\mathsf{entry}_{\mathrm{I}}(\mathsf{A}),\mathsf{sub}_{\mathrm{I}}(\mathsf{A}))\wedge\mathsf{index}_{\mathrm{I}}(\mathsf{A})\notin_{\mathrm{I}}\mathsf{sub}_{\mathrm{I}}(\mathsf{A}))\\ \qquad\rightarrow(\mathsf{false}\vee\Delta^{\mathrm{I}1}_{\mathsf{sub}_{\mathrm{I}}}(\mathsf{A}))\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{array}_{\mathrm{I}}\\ \quad(\mathsf{A}\equiv\mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}),\mathsf{entry}_{\mathrm{I}}(\mathsf{A}),\mathsf{sub}_{\mathrm{I}}(\mathsf{A}))\wedge\mathsf{index}_{\mathrm{I}}(\mathsf{A})\notin_{\mathrm{I}}\mathsf{sub}_{\mathrm{I}}(\mathsf{A}))\\ \qquad\rightarrow\mathsf{A}\equiv\mathsf{put}_{\mathrm{I}}(\mathsf{index}_{\mathrm{I}}(\mathsf{A}),\mathsf{entry}_{\mathrm{I}}(\mathsf{A}),\mathsf{sub}_{\mathrm{I}}(\mathsf{A})).\end{array}$$

Now, we need to prove that the above axiomatization of $\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}$ computes the minimal size of an array, indeed. Therefore we need to show the following proof obligations

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{array}_{\mathrm{I}}\\ \quad\mathsf{Eq}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A},\mathsf{B})\equiv\mathsf{true}\rightarrow(\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A})\leq_{\mathsf{nat}}\mathsf{term\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{B}))\equiv\mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A}:\mathsf{array}_{\mathrm{I}}\ \exists\ \mathsf{B}:\mathsf{array}_{\mathrm{I}}\\ \quad\mathsf{Eq}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A},\mathsf{B})\equiv\mathsf{true}\wedge(\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A})\geq_{\mathsf{nat}}\mathsf{term\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{B}))\equiv\mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{A},\mathsf{B}:\mathsf{array}_{\mathrm{I}}\\ \quad\mathsf{Eq}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A},\mathsf{B})\equiv\mathsf{true}\rightarrow\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A})\equiv\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{B})\end{array}$$

Next, we need to show that put denotes a size increasing constructor function. To do that, we prove:

$$\begin{array}{l}\forall\ \mathsf{i}:\mathsf{nat}\ \forall\ \mathsf{x}:\mathsf{nat}\ \forall\ \mathsf{A}:\mathsf{array}_{\mathrm{I}}\\ \quad(\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A})\leq_{\mathsf{nat}}\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{put}_{\mathrm{I}}(\mathsf{i},\mathsf{x},\mathsf{A})))\equiv\mathsf{true}.\end{array}$$

Finally, we need to define the strictness predicate $\Theta^3_{\mathsf{put}_{\mathrm{I}}}:\mathsf{nat}\times\mathsf{array}_{\mathrm{I}}\rightarrow\mathsf{bool}$ and the minimal representation predicate $\Gamma_{\mathsf{put}_{\mathrm{I}}}:\mathsf{nat}\times\mathsf{array}_{\mathrm{I}}\rightarrow\mathsf{bool}$. We suggest the following definitions:

$$\begin{array}{l}\forall\ \mathsf{i}:\forall\ \ :\mathsf{x}\ \mathsf{nat}\forall\ \mathsf{A}:\mathsf{array}_{\mathrm{I}}\\ \quad\Theta^3_{\mathsf{put}_{\mathrm{I}}}(\mathsf{i},\mathsf{x},\mathsf{A})\equiv\mathsf{true}\\ \qquad\leftrightarrow\mathsf{i}\notin_{\mathrm{I}}\mathsf{A}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{i}:\forall\ \ :\mathsf{x}\ \mathsf{nat}\forall\ \mathsf{A}:\mathsf{array}_{\mathrm{I}}\\ \quad\Gamma_{\mathsf{put}_{\mathrm{I}}}(\mathsf{i},\mathsf{x},\mathsf{A})\equiv\mathsf{true}\\ \qquad\leftrightarrow\mathsf{i}\notin_{\mathrm{I}}\mathsf{A}\end{array}$$

However, we have to prove that our suggestions really define the strictness and the minimal representation predicate. Hence, we need to show that

$$\begin{array}{l}\forall\ \mathsf{i}:\forall\ \ :\mathsf{x}\ \mathsf{nat}\forall\ \mathsf{A}:\mathsf{array}_{\mathrm{I}}\\ \quad\Theta^3_{\mathsf{put}_{\mathrm{I}}}(\mathsf{i},\mathsf{x},\mathsf{A})\equiv\mathsf{true}\\ \qquad\leftrightarrow(\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A})<_{\mathsf{nat}}\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{put}_{\mathrm{I}}(\mathsf{i},\mathsf{x},\mathsf{A})))\equiv\mathsf{true}\end{array}$$

$$\begin{array}{l}\forall\ \mathsf{i}:\forall\ \ :\mathsf{x}\ \mathsf{nat}\forall\ \mathsf{A}:\mathsf{array}_{\mathrm{I}}\\ \quad\Gamma_{\mathsf{put}_{\mathrm{I}}}(\mathsf{i},\mathsf{x},\mathsf{A})\equiv\mathsf{true}\\ \qquad\leftrightarrow\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{put}_{\mathrm{I}}(\mathsf{i},\mathsf{x},\mathsf{A}))\equiv\mathsf{succ}(\mathsf{min\_size}_{\mathsf{array}_{\mathrm{I}}}(\mathsf{A}))\end{array}$$

Having done so, we know for our original specification array that the constructor function put is size increasing, and we can translate the strictness predicate as well as the minimal representation predicate into the original specification. Hence, we obtain:

$$\begin{array}{l}\forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array}\\ \quad \Theta^3_{\mathsf{put}}(\mathsf{i},\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{i} \notin \mathsf{A}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array}\\ \quad \Gamma_{\mathsf{put}}(\mathsf{i},\mathsf{x},\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{i} \notin \mathsf{A}\end{array}$$

The data type array possesses non-overlapping constructor functions, since

$$\begin{array}{l}\forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array}\\ \quad \mathsf{void} \not\equiv \mathsf{put}(\mathsf{i},\mathsf{x},\mathsf{A})\end{array}$$

holds. Hence, we can use the simplified construction scheme for the destructor functions.

For the constructor function put we introduce three destructor functions index : array $\rightarrow$ nat for the first argument of put, entry : array $\rightarrow$ nat for the second argument of put and sub : array $\rightarrow$ array for the third argument of put. For these destructor functions we obtain the following representation axioms:

$$\begin{array}{l}\forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}:\mathsf{array}\\ \quad \mathsf{A} \equiv \mathsf{put}(\mathsf{i},\mathsf{x},\mathsf{B}) \rightarrow \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{B}))\end{array}$$

$$\mathsf{index}(\mathsf{void}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\mathsf{entry}(\mathsf{void}) \equiv 0\ (\equiv \bigtriangledown_{\mathsf{nat}})$$

$$\mathsf{sub}(\mathsf{void}) \equiv \mathsf{void}$$

$$\begin{array}{l}\forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{x}:\mathsf{nat}\ \forall\, \mathsf{A},\mathsf{B}:\mathsf{array}\\ \quad \mathsf{A} \equiv \mathsf{put}(\mathsf{i},\mathsf{x},\mathsf{B})\\ \qquad \rightarrow \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}\end{array}$$

The reflexive destructor function of the constructor function put, sub, is 1-bounded, and the difference predicate Δ^1_{sub} : array $\rightarrow$ bool is defined by

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{array}\\ \quad \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true}\\ \qquad \leftrightarrow \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\end{array}$$

For the data type array we will give constructive function and predicate specifications for delete, size, min_index, index_min, swap, sort, $<_{\mathsf{array}}$, $\leq_{\mathsf{array}}$, $>_{\mathsf{array}}$, and $\geq_{\mathsf{array}}$.

12.1 delete : nat × array → array

delete removes the entry at the specified index from an array. It is defined by:

$$\begin{array}{l}\forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array}\\ \quad \mathsf{A} \equiv \mathsf{void} \rightarrow \mathsf{delete}(\mathsf{i},\mathsf{A}) \equiv \mathsf{void}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array}\\ \quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \equiv \mathsf{index}(\mathsf{A}))\\ \qquad \rightarrow \mathsf{delete}(\mathsf{i},\mathsf{A}) \equiv \mathsf{sub}(\mathsf{A})\end{array}$$

$$\begin{aligned}
&\forall\, \mathsf{i}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{array}\\
&\quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \not\equiv \mathsf{index}(\mathsf{A}))\\
&\qquad \to \mathsf{delete}(\mathsf{i}, \mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{delete}(\mathsf{i}, \mathsf{sub}(\mathsf{A})))
\end{aligned}$$

The recursion ordering of delete is well-founded. There is only a single recursive definition case with a single recursive call. Hence, we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \not\equiv \mathsf{index}(\mathsf{A}))$$

by φ. Using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove:

$$\begin{aligned}
&\forall\, \mathsf{i}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{array}\\
&\quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \not\equiv \mathsf{index}(\mathsf{A}))\\
&\qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true})
\end{aligned}$$

which simplifies to

$$\begin{aligned}
&\forall\, \mathsf{i}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{array}\\
&\quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \not\equiv \mathsf{index}(\mathsf{A}))\\
&\qquad \to \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))
\end{aligned}$$

In addition, delete denotes a 2-bounded function symbol. First of all, we prove that delete is completely specified, by

$$\begin{aligned}
&\forall\, \mathsf{i}{:}\mathsf{nat}\ \forall\, \mathsf{A}{:}\mathsf{array}\\
&\qquad\qquad\qquad \mathsf{A} \equiv \mathsf{void} \vee\\
&\quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \equiv \mathsf{index}(\mathsf{A})) \vee\\
&\quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \not\equiv \mathsf{index}(\mathsf{A}))
\end{aligned}$$

Next, we examine each definition case separately. For the first case we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{void}$$

by φ. Using the Estimation Calculus, we obtain:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{void} \preceq_{\mathsf{array}} \mathsf{void}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{void} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Equation\ 1}$$

For the second case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \equiv \mathsf{index}(\mathsf{A}))$$

by φ. Thus, we obtain the derivation

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

For the third case we abbreviate the invariant case condition

$$(\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \not\equiv \mathsf{index}(\mathsf{A}))$$

by φ. Since this is a recursive case, we may assume the additional inference rule

$$\xi \Rightarrow \dfrac{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \Delta\rangle}{\langle \varphi, \mathsf{delete}(\mathsf{i}, \mathsf{sub}(\mathsf{A})) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{i}, \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \mathsf{Induction\ Hypothesis}$$

as induction hypothesis, where ξ is an abbreviation for the formula

$$\forall\, \mathsf{i}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\ \varphi \to \Delta$$

Then, we obtain

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}}{\langle \varphi, \mathsf{delete}(\mathsf{i}, \mathsf{sub}(\mathsf{A})) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \Delta^2_{\mathsf{delete}}(\mathsf{i}, \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \mathsf{Induction\ Hypothesis}}{\left\langle \begin{array}{c} \varphi, \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{delete}(\mathsf{i}, \mathsf{sub}(\mathsf{A}))) \\ \preceq_{\mathsf{array}} \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})), \\ \Delta^2_{\mathsf{delete}}(\mathsf{i}, \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{delete}(\mathsf{i}, \mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \mathsf{Weak\ Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{delete}(\mathsf{i}, \mathsf{sub}(\mathsf{A}))) \preceq_{\mathsf{array}} \mathsf{A}, \\ \Delta^2_{\mathsf{delete}}(\mathsf{i}, \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{delete}(\mathsf{i}, \mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \mathsf{Equation\ 3}$$

where in order to apply the induction hypothesis, the formula

$$\forall\, \mathsf{i}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array}\ \varphi \to \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}$$

needs to be proved.

The difference predicate $\Delta^2_{\mathsf{delete}} : \mathsf{nat} \times \mathsf{array} \to \mathsf{bool}$ is then synthesized, using the simplified difference formulas from each derivation as

$$\begin{array}{l} \forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array} \\ \quad \mathsf{A} \equiv \mathsf{void} \to \Delta^2_{\mathsf{delete}}(\mathsf{i}, \mathsf{A}) \equiv \mathsf{false} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array} \\ \quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \equiv \mathsf{index}(\mathsf{A})) \\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{i}, \mathsf{A}) \equiv \mathsf{true} \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array} \\ \quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \mathsf{i} \not\equiv \mathsf{index}(\mathsf{A})) \\ \qquad \to \Delta^2_{\mathsf{delete}}(\mathsf{i}, \mathsf{A}) \equiv \Delta^2_{\mathsf{delete}}(\mathsf{i}, \mathsf{sub}(\mathsf{A})) \end{array}$$

12.2 $\mathsf{size} : \mathsf{array} \to \mathsf{nat}$

size computes the number of occupied entries in an array, and it is defined by:

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{array} \\ \quad \mathsf{A} \equiv \mathsf{void} \to \mathsf{size}(\mathsf{A}) \equiv 0 \end{array}$$

$$\begin{array}{l} \forall\, \mathsf{A}:\mathsf{array} \\ \quad \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \\ \qquad \to \mathsf{size}(\mathsf{A}) \equiv \mathsf{succ}(\mathsf{size}(\mathsf{sub}(\mathsf{A}))) \end{array}$$

The recursion ordering of size is well-founded. There is only a single recursive definition case with a single recursive call. Hence, we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))$$

by φ. Using the Estimation Calculus, we obtain the derivation:

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\ \mathsf{Estimation}$$

To ensure the strict relation, we need to prove:

$$\begin{array}{l} \forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array} \\ \quad \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \\ \qquad \to (\mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which simplifies to

$$\begin{array}{l} \forall\, \mathsf{i}:\mathsf{nat}\ \forall\, \mathsf{A}:\mathsf{array} \\ \quad \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \\ \qquad \to \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \end{array}$$

12.3 min_index : array → nat

min_index computes the minimal index in an array. It is defined by:

$$\forall\, \mathsf{A}{:}\mathsf{array}\ (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \mathsf{sub}(\mathsf{A}) \equiv \mathsf{void}) \rightarrow \mathsf{min\_index}(\mathsf{A}) \equiv \mathsf{index}(\mathsf{A})$$

$$\forall\, \mathsf{A}{:}\mathsf{array} \begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})),\mathsf{entry}(\mathsf{sub}(\mathsf{A})),\mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge \\ (\mathsf{index}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{index}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix} \rightarrow \mathsf{min\_index}(\mathsf{A}) \equiv \mathsf{min\_index}(\mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))))$$

$$\forall\, \mathsf{A}{:}\mathsf{array} \begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})),\mathsf{entry}(\mathsf{sub}(\mathsf{A})),\mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge \\ (\mathsf{index}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{index}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{pmatrix} \rightarrow \mathsf{min\_index}(\mathsf{A}) \equiv \mathsf{min\_index}(\mathsf{sub}(\mathsf{A}))$$

The recursion ordering of min_index is well-founded. There are two recursive definition cases with a single recursive call in each. For the first case we abbreviate the invariant case condition

$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})),\mathsf{entry}(\mathsf{sub}(\mathsf{A})),\mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge \\ (\mathsf{index}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{index}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true} \end{pmatrix}$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{sub}(\mathsf{A})) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}}{\left\langle \begin{matrix} \varphi, \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \\ \preceq_{\mathsf{array}} \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})), \\ \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{matrix} \right\rangle}\ \mathsf{Weak\ Embedding}}{\left\langle \begin{matrix} \varphi, \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \preceq_{\mathsf{array}} \mathsf{A}, \\ \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{matrix} \right\rangle}\ \mathsf{Equation\ 3}$$

where to enable the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}{:}\mathsf{array}\ \varphi \rightarrow \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}$$

has to be proved. To ensure the strict relation, we need to show

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{array} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge \\ (\mathsf{index}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{index}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right) \\ \quad\quad \to \left(\begin{array}{c}\mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}\vee \\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{array} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge \\ (\mathsf{index}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{index}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right) \\ \quad\quad \to \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A})))\end{array}$$

For the second case we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge \\ (\mathsf{index}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{index}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true}\rangle}\ \mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{array} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge \\ (\mathsf{index}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{index}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right) \\ \quad\quad \to (\mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true})\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}\!:\!\mathsf{array} \\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge \\ (\mathsf{index}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{index}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right) \\ \quad\quad \to \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))\end{array}$$

12.4 index_min : array → nat

index_min computes the index for the minimal entry in an array. It is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{array}\\ \quad (\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \mathsf{sub}(\mathsf{A}) \equiv \mathsf{void})\\ \qquad \to \mathsf{index\_min}(\mathsf{A}) \equiv \mathsf{index}(\mathsf{A})\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{array}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge\\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})),\mathsf{entry}(\mathsf{sub}(\mathsf{A})),\mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge\\ (\mathsf{entry}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{entry}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right)\\ \qquad \to \mathsf{index\_min}(\mathsf{A}) \equiv \mathsf{index\_min}(\mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))))\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{A}{:}\mathsf{array}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge\\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})),\mathsf{entry}(\mathsf{sub}(\mathsf{A})),\mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge\\ (\mathsf{entry}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{entry}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right)\\ \qquad \to \mathsf{index\_min}(\mathsf{A}) \equiv \mathsf{index\_min}(\mathsf{sub}(\mathsf{A}))\end{array}$$

The recursion ordering of index_min is well-founded. There are two recursive definition cases with a single recursive call in each. For the first case we abbreviate the invariant case condition

$$\left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge\\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})),\mathsf{entry}(\mathsf{sub}(\mathsf{A})),\mathsf{sub}(\mathsf{sub}(\mathsf{A})))\wedge\\ (\mathsf{entry}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{entry}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true}\end{array}\right)$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\begin{array}{c}-\\ \hline \text{Identity}\\ \langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \mathsf{false}\rangle\\ \hline \text{Estimation}\\ \langle \varphi, \mathsf{sub}(\mathsf{sub}(\mathsf{A})) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}\rangle\\ \hline \text{Weak Embedding}\\ \left\langle \begin{array}{c}\varphi, \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A})))\\ \preceq_{\mathsf{array}} \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})),\\ \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right\rangle\\ \hline \text{Equation 3}\\ \left\langle \begin{array}{c}\varphi, \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \preceq_{\mathsf{array}} \mathsf{A},\\ \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee\\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false}\end{array}\right\rangle\end{array}$$

where to enable the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{A}{:}\mathsf{array}\ \varphi \to \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}$$

has to be proved. To ensure the strict relation, we need to show

$$\forall\, \mathsf{A}\!:\!\mathsf{array}\\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \wedge \\ (\mathsf{entry}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{entry}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true} \end{array}\right) \\ \rightarrow \left(\begin{array}{c} \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array}\right)$$

which can be simplified to

$$\forall\, \mathsf{A}\!:\!\mathsf{array}\\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \wedge \\ (\mathsf{entry}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{entry}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true} \end{array}\right) \\ \rightarrow \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A})))$$

For the second case we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \wedge \\ (\mathsf{entry}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{entry}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array}\right)$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \rangle}\,\mathsf{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\,\mathsf{Estimation}$$

To prove the strict relation, we need to show

$$\forall\, \mathsf{A}\!:\!\mathsf{array}\\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \wedge \\ (\mathsf{entry}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{entry}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true} \end{array}\right) \\ \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true})$$

which can be simplified to

$$\forall\, \mathsf{A}\!:\!\mathsf{array}\\ \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{sub}(\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{sub}(\mathsf{A})), \mathsf{entry}(\mathsf{sub}(\mathsf{A})), \mathsf{sub}(\mathsf{sub}(\mathsf{A}))) \wedge \\ (\mathsf{entry}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{entry}(\mathsf{sub}(\mathsf{A}))) \equiv \mathsf{true} \end{array}\right) \\ \rightarrow \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))$$

12.5 swap : nat × nat × array → nat

swap swaps the entries in an array at the specified indices. It is defined by:

$$\forall\, \mathsf{i},\mathsf{j}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\quad \mathsf{A} \equiv \mathsf{void} \to \mathsf{swap}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{A}$$

$$\forall\, \mathsf{i},\mathsf{j}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{i} \notin \mathsf{A} \end{pmatrix} \to \mathsf{swap}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{A}$$

$$\forall\, \mathsf{i},\mathsf{j}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{i} \in \mathsf{A}\wedge \\ \mathsf{j} \notin \mathsf{A} \end{pmatrix} \to \mathsf{swap}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{A}$$

$$\forall\, \mathsf{i},\mathsf{j}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{i} \in \mathsf{A}\wedge \\ \mathsf{j} \in \mathsf{A}\wedge \\ \mathsf{i} \equiv \mathsf{j} \end{pmatrix} \to \mathsf{swap}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{A}$$

$$\forall\, \mathsf{i},\mathsf{j}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{i} \in \mathsf{A}\wedge \\ \mathsf{j} \in \mathsf{A}\wedge \\ \mathsf{i} \not\equiv \mathsf{j}\wedge \\ \mathsf{index}(\mathsf{A}) \equiv \mathsf{i} \end{pmatrix} \to \mathsf{swap}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{put}(\mathsf{i},\mathsf{aref}(\mathsf{j},\mathsf{sub}(\mathsf{A})),\mathsf{put}(\mathsf{j},\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})))$$

$$\forall\, \mathsf{i},\mathsf{j}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{i} \in \mathsf{A}\wedge \\ \mathsf{j} \in \mathsf{A}\wedge \\ \mathsf{i} \not\equiv \mathsf{j}\wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i}\wedge \\ \mathsf{index}(\mathsf{A}) \equiv \mathsf{j} \end{pmatrix} \to \mathsf{swap}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{put}(\mathsf{j},\mathsf{aref}(\mathsf{i},\mathsf{sub}(\mathsf{A})),\mathsf{put}(\mathsf{i},\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})))$$

$$\forall\, \mathsf{i},\mathsf{j}\colon \mathsf{nat}\ \forall\, \mathsf{A}\colon \mathsf{array}\quad \begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge \\ \mathsf{i} \in \mathsf{A}\wedge \\ \mathsf{j} \in \mathsf{A}\wedge \\ \mathsf{i} \not\equiv \mathsf{j}\wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i}\wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{j} \end{pmatrix} \to \mathsf{swap}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{swap}(\mathsf{i},\mathsf{j},\mathsf{sub}(\mathsf{A})))$$

The recursion ordering of swap is well-founded. There is only a single recursive definition case with a single recursive call. We abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{j} \end{array}\right)$$

$$\frac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \rangle}\ \text{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\ \text{Estimation}$$

To ensure the strict relation, we need to prove:

$$\begin{array}{l} \forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{j} \end{array}\right) \\ \quad \rightarrow (\mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true}) \end{array}$$

which simplifies to

$$\begin{array}{l} \forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array} \\ \quad \left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{j} \end{array}\right) \\ \quad \rightarrow \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \end{array}$$

In addition, swap denotes a 3-bounded function symbol. To prove that, first of all, we show that swap is completely specified, by

$$\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}$$

$$\begin{array}{c}
\mathsf{A} \equiv \mathsf{void} \vee \\
\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \notin \mathsf{A} \end{array}\right) \vee \\
\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \notin \mathsf{A} \end{array}\right) \vee \\
\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \equiv \mathsf{j} \end{array}\right) \vee \\
\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \equiv \mathsf{i} \end{array}\right) \vee \\
\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i} \wedge \\ \mathsf{index}(\mathsf{A}) \equiv \mathsf{j} \end{array}\right) \vee \\
\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{j} \end{array}\right)
\end{array}$$

Next, we examine each definition case separately. For the first, second, third, and fourth definition case although we have different case conditions, abbreviated by φ, we have identical derivations in the Estimation Calculus:

$$\frac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false}\rangle}\ \mathsf{Identity}$$

For the fifth definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \equiv \mathsf{i} \end{array}\right)$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{put}(\mathsf{j}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \mathsf{false}\rangle}\ \text{Strict Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{put}(\mathsf{i}, \mathsf{aref}(\mathsf{j}, \mathsf{sub}(\mathsf{A})), \mathsf{put}(\mathsf{j}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))) \\ \preceq_{\mathsf{array}} \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})), \\ \mathsf{false} \vee \Gamma_{\mathsf{put}}(\mathsf{i}, \mathsf{aref}(\mathsf{j}, \mathsf{sub}(\mathsf{A})), \mathsf{put}(\mathsf{j}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{put}(\mathsf{i}, \mathsf{aref}(\mathsf{j}, \mathsf{sub}(\mathsf{A})), \mathsf{put}(\mathsf{j}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))) \preceq_{\mathsf{array}} \mathsf{A}, \\ \mathsf{false} \vee \Gamma_{\mathsf{put}}(\mathsf{i}, \mathsf{aref}(\mathsf{j}, \mathsf{sub}(\mathsf{A})), \mathsf{put}(\mathsf{j}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where to enable the application of the Strict Embedding Rule, the formula

$$\forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array}\ \varphi \to \Theta^3_{\mathsf{put}}(\mathsf{j}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \equiv \mathsf{false}$$

has to be proved, and where to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array}\ \varphi \to \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}$$

needs to be shown. For the sixth definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i} \wedge \\ \mathsf{index}(\mathsf{A}) \equiv \mathsf{j} \end{array} \right)$$

by φ. Using the Estimation Calculus, we obtain the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \mathsf{false}\rangle}\ \text{Identity}}{\langle \varphi, \mathsf{put}(\mathsf{i}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \mathsf{false}\rangle}\ \text{Strict Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{put}(\mathsf{j}, \mathsf{aref}(\mathsf{i}, \mathsf{sub}(\mathsf{A})), \mathsf{put}(\mathsf{i}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))) \\ \preceq_{\mathsf{array}} \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})), \\ \mathsf{false} \vee \Gamma_{\mathsf{put}}(\mathsf{j}, \mathsf{aref}(\mathsf{i}, \mathsf{sub}(\mathsf{A})), \mathsf{put}(\mathsf{i}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{put}(\mathsf{j}, \mathsf{aref}(\mathsf{i}, \mathsf{sub}(\mathsf{A})), \mathsf{put}(\mathsf{i}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))) \preceq_{\mathsf{array}} \mathsf{A}, \\ \mathsf{false} \vee \Gamma_{\mathsf{put}}(\mathsf{j}, \mathsf{aref}(\mathsf{i}, \mathsf{sub}(\mathsf{A})), \mathsf{put}(\mathsf{i}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where to enable the application of the Strict Embedding Rule, the formula

$$\forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array}\ \varphi \rightarrow \Theta^3_{\mathsf{put}}(\mathsf{i}, \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \equiv \mathsf{false}$$

has to be proved, and where to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array}\ \varphi \rightarrow \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}$$

needs to be shown. For the seventh definition case we abbreviate the invariant case condition

$$\left(\begin{array}{c} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \wedge \\ \mathsf{i} \in \mathsf{A} \wedge \\ \mathsf{j} \in \mathsf{A} \wedge \\ \mathsf{i} \not\equiv \mathsf{j} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i} \wedge \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{j} \end{array} \right)$$

by φ. Since this is a recursive case, we may assume the additional inference rule

$$\xi \Rightarrow \frac{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \Delta \rangle}{\left\langle \varphi, \mathsf{swap}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A})) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \Delta^3_{\mathsf{swap}}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \right\rangle} \text{ Induction Hypothesis}$$

as an induction hypothesis, where ξ is an abbreviation for the formula

$$\forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array}\ \varphi \rightarrow \Delta$$

Thus, we obtain the derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \rangle}\ \text{Identity}}{\langle \varphi, \mathsf{sub}(\mathsf{A}) \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true} \rangle}\ \text{Estimation}}{\left\langle \begin{array}{c} \varphi, \mathsf{swap}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A})) \preceq_{\mathsf{array}} \mathsf{sub}(\mathsf{A}), \\ \Delta^3_{\mathsf{swap}}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Induction Hypothesis}}{\left\langle \begin{array}{c} \varphi, \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{swap}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A}))) \\ \preceq_{\mathsf{array}} \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})), \\ \Delta^3_{\mathsf{swap}}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{swap}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Weak Embedding}}{\left\langle \begin{array}{c} \varphi, \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{swap}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A}))) \preceq_{\mathsf{array}} \mathsf{A}, \\ \Delta^3_{\mathsf{swap}}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A})) \equiv \mathsf{true} \vee \\ \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{swap}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A}))) \equiv \mathsf{false} \end{array} \right\rangle}\ \text{Equation 3}$$

where in order to enable the application of the induction hypothesis, the formula

$$\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\ \varphi \to (\mathsf{false} \vee \Delta^1_{\mathsf{sub}}(\mathsf{A}) \equiv \mathsf{true})$$

has to be proved, and to allow the application of the Weak Embedding Rule, the formula

$$\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\ \varphi \to \Gamma_{\mathsf{put}}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A})) \equiv \mathsf{true}$$

needs to be shown.

Now, we can synthesize the difference predicate $\Delta^3_{\mathsf{swap}} : \mathsf{nat} \times \mathsf{nat} \times \mathsf{array} \to \mathsf{bool}$, using the simplified difference formulas from each derivation in the Estimation Calculus, as

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\\ \quad \mathsf{A} \equiv \mathsf{void} \to \Delta^3_{\mathsf{swap}}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge\\ \mathsf{i} \notin \mathsf{A}\end{array}\right)\\ \quad\ \to \Delta^3_{\mathsf{swap}}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge\\ \mathsf{i} \in \mathsf{A}\wedge\\ \mathsf{j} \notin \mathsf{A}\end{array}\right)\\ \quad\ \to \Delta^3_{\mathsf{swap}}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge\\ \mathsf{i} \in \mathsf{A}\wedge\\ \mathsf{j} \in \mathsf{A}\wedge\\ \mathsf{i} \equiv \mathsf{j}\end{array}\right)\\ \quad\ \to \Delta^3_{\mathsf{swap}}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge\\ \mathsf{i} \in \mathsf{A}\wedge\\ \mathsf{j} \in \mathsf{A}\wedge\\ \mathsf{i} \not\equiv \mathsf{j}\wedge\\ \mathsf{index}(\mathsf{A}) \equiv \mathsf{i}\end{array}\right)\\ \quad\ \to \Delta^3_{\mathsf{swap}}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\begin{array}{l}\forall\, \mathsf{i},\mathsf{j}\!:\!\mathsf{nat}\ \forall\, \mathsf{A}\!:\!\mathsf{array}\\ \quad \left(\begin{array}{c}\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}),\mathsf{entry}(\mathsf{A}),\mathsf{sub}(\mathsf{A}))\wedge\\ \mathsf{i} \in \mathsf{A}\wedge\\ \mathsf{j} \in \mathsf{A}\wedge\\ \mathsf{i} \not\equiv \mathsf{j}\wedge\\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i}\wedge\\ \mathsf{index}(\mathsf{A}) \equiv \mathsf{j}\end{array}\right)\\ \quad\ \to \Delta^3_{\mathsf{swap}}(\mathsf{i},\mathsf{j},\mathsf{A}) \equiv \mathsf{false}\end{array}$$

$$\forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array}$$
$$\begin{pmatrix} \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A})) \land \\ \mathsf{i} \in \mathsf{A} \land \\ \mathsf{j} \in \mathsf{A} \land \\ \mathsf{i} \not\equiv \mathsf{j} \land \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{i} \land \\ \mathsf{index}(\mathsf{A}) \not\equiv \mathsf{j} \end{pmatrix}$$
$$\rightarrow \Delta^3_{\mathsf{swap}}(\mathsf{i}, \mathsf{j}, \mathsf{A}) \equiv \Delta^3_{\mathsf{swap}}(\mathsf{i}, \mathsf{j}, \mathsf{sub}(\mathsf{A}))$$

which can be further simplified to

$$\forall\, \mathsf{i}, \mathsf{j} : \mathsf{nat}\ \forall\, \mathsf{A} : \mathsf{array}$$
$$\Delta^3_{\mathsf{swap}}(\mathsf{i}, \mathsf{j}, \mathsf{A}) \equiv \mathsf{false}$$

12.6 sort : array → nat

sort sorts an array, and it is defined by:

$$\forall\, \mathsf{A} : \mathsf{array}$$
$$\mathsf{A} \equiv \mathsf{void} \rightarrow \mathsf{sort}(\mathsf{A}) \equiv \mathsf{void}$$

$$\forall\, \mathsf{A} : \mathsf{array}$$
$$\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))$$
$$\rightarrow \mathsf{sort}(\mathsf{A}) \equiv$$
$$\mathsf{put}(\mathsf{min\_index}(\mathsf{A}), \mathsf{aref}(\mathsf{index\_min}(\mathsf{A}), \mathsf{A}),$$
$$\mathsf{sort}(\mathsf{delete}(\mathsf{min\_index}(\mathsf{A}),$$
$$\mathsf{swap}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A}))))$$

The recursion ordering of size is well-founded. There is only a single recursive definition case with a single recursive call. Hence, we abbreviate the invariant case condition

$$\mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))$$

by φ. Using the Estimation Calculus, we obtain the derivation:

$$\cfrac{\cfrac{\cfrac{-}{\langle \varphi, \mathsf{A} \preceq_{\mathsf{array}} \mathsf{A}, \mathsf{false} \rangle}\ \text{Identity}}{\left\langle \begin{array}{c} \varphi, \mathsf{swap}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A})) \preceq_{\mathsf{array}} \mathsf{A}, \\ \mathsf{false} \lor \Delta^3_{\mathsf{swap}}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Estimation}}{\left\langle \begin{array}{c} \varphi, \mathsf{delete}(\mathsf{min\_index}(\mathsf{A}), \mathsf{swap}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A})) \preceq_{\mathsf{array}} \mathsf{A}, \\ \mathsf{false} \lor \Delta^3_{\mathsf{swap}}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A})) \equiv \mathsf{true} \\ \Delta^2_{\mathsf{delete}}(\mathsf{min\_index}(\mathsf{A}), \mathsf{swap}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A})) \equiv \mathsf{true} \end{array} \right\rangle}\ \text{Estimation}$$

To ensure the strict relation we must prove

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{array}\\ \quad \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))\\ \qquad \rightarrow \begin{pmatrix} \mathsf{false} \vee \Delta^3_{\mathsf{swap}}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A})) \equiv \mathsf{true} \\ \Delta^2_{\mathsf{delete}}(\mathsf{min\_index}(\mathsf{A}), \mathsf{swap}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A})) \equiv \mathsf{true}\end{pmatrix}\end{array}$$

which can be simplified to

$$\begin{array}{l}\forall\, \mathsf{A}:\mathsf{array}\\ \quad \mathsf{A} \equiv \mathsf{put}(\mathsf{index}(\mathsf{A}), \mathsf{entry}(\mathsf{A}), \mathsf{sub}(\mathsf{A}))\\ \qquad \rightarrow \Delta^2_{\mathsf{delete}}(\mathsf{min\_index}(\mathsf{A}), \mathsf{swap}(\mathsf{min\_index}(\mathsf{A}), \mathsf{index\_min}(\mathsf{A}), \mathsf{A})) \equiv \mathsf{true}\end{array}$$

whose proof can be done by induction.

12.7 $<_{\mathsf{array}}$: array × array → bool

$<_{\mathsf{array}}$ computes the less-than-relation on arrays and is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}:\mathsf{array}\\ \quad (\mathsf{A} <_{\mathsf{array}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{size}(\mathsf{A}) <_{\mathsf{nat}} \mathsf{size}(\mathsf{B})) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive specification, we are done. However note, $<_{\mathsf{array}}$ denotes a well-founded order relation.

12.8 $\leq_{\mathsf{array}}$: array × array → bool

$\leq_{\mathsf{array}}$ computes the less-than-or-equal-relation on arrays and is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}:\mathsf{array}\\ \quad (\mathsf{A} \leq_{\mathsf{array}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{size}(\mathsf{A}) \leq_{\mathsf{nat}} \mathsf{size}(\mathsf{A})) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive specification, we are done.

12.9 $>_{\mathsf{array}}$: array × array → bool

$>_{\mathsf{array}}$ computes the greater-than-relation on arrays and is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}:\mathsf{array}\\ \quad (\mathsf{A} >_{\mathsf{array}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} <_{\mathsf{array}} \mathsf{A}) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive specification, we are done.

12.10 $\geq_{\mathsf{array}}$: array × array → bool

$\geq_{\mathsf{array}}$ computes the greater-than-or-equal-relation on arrays and is defined by:

$$\begin{array}{l}\forall\, \mathsf{A}, \mathsf{B}:\mathsf{array}\\ \quad (\mathsf{A} \geq_{\mathsf{array}} \mathsf{B}) \equiv \mathsf{true} \leftrightarrow (\mathsf{B} \leq_{\mathsf{array}} \mathsf{A}) \equiv \mathsf{true}\end{array}$$

Since this is a non-recursive specification, we are done.