Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-93-12

RELFUN Guide: Programming with Relations and Functions Made Easy

(Second, Revised Edition)

Harold Boley,
Simone Andel, Klaus Elsbernd, Michael Herfert,
Michael Sintek, Werner Stein

July 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

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ☐ Intelligent Engineering Systems
- ☐ Intelligent User Interfaces
- ☐ Computer Linguistics
- ☐ Programming Systems
- ☐ Deduction and Multiagent Systems
- ☐ Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

RELFUN Guide: Programming with Relations and Functions Made Easy

Harold Boley,
Simone Andel, Klaus Elsbernd, Michael Herfert,
Michael Sintek, Werner Stein

DFKI-D-93-12

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITW-8902 C4 and 413-5839-ITW-9304/3).

ISSN 0946-0098

RELFUN Guide:
Programming with Relations and Functions Made Easy

Harold Boley,
Simone Andel, Klaus Elsbernd, Michael Herfert,
Michael Sintek, Werner Stein

DFKI
Universität Kaiserslautern
Erwin-Schrödinger-Straße
67663 Kaiserslautern
Germany

Second, Revised Edition

July 1996

Abstract

A practical description of relational/functional programming in **RELFUN** is given. The language constructs are introduced by a tutorial dialog. Builtins, primitives, and commands are explained. Examples are given on all aspects relevant to using the language.

Contents

12 June 1996

GENERAL INFORMATION ON THE RELFUN SYSTEM
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

CONCEPTS:
RELFUN is a logic-programming language with call-by-value (eager) expressions of non-deterministic, non-ground functions [Boley92b]. Clauses can be 'hornish', succeeding with true(s), and 'footed', returning any value(s). This is the only remaining difference between relations and functions, uniformly considered as operators. RELFUN allows (apply-reducible) higher-order notation with arbitrary terms (constants, structures, and variables) in operator positions of clause heads and bodies. (Active) expressions, i.e. "(...)"-applications of defined operators, are explicitly distinguished from (passive) structures, i.e. "[...]"- applications of constructors. PROLOG's is-call is generalized to an equational goal unifying a term with the value(s) of an arbitrary expression, which may become flattened via further is-calls. All structures and expressions, not only lists, can be "|"-unified polyadically. Finite domains/exclusions [Boley93] and "$"-prefixed predicates-as-sorts (builtin or user-defined [Hall95]) are first-class terms handled by unification. RELFUN extensions include single-cut clauses and relational-functional primitives such as a value-returning tupof.

SEMANTICS, IMPLEMENTION, INTERACTION, APPLICATIONS:
Operational (interpreter in pure LISP), procedural (SLV-resolution), fixpoint, and model-theoretic semantics for pure RELFUN [Boley92c].
Interpreter for full RELFUN; WAM compiler/emulator almost identical.
Compiler system layered, from source-to-source transformers to declarative classifier, to WAM-code generator [Boley90]/[Sintek95]/[BoleyElsbernd+96].
Translator to relational subset of RELFUN and partially to PROLOG.
Accepting freely interchangeable LISP-style and PROLOG-style syntaxes.
Module system in analogy to file system.
Spier for valued conjunctions; generalized box-model tracer. On-line help.
Interface to (used for builtins) and from LISP.
Prelude with useful relations/functions; library of declarative hypergraph operations [Boley92a]; components for mechanical-engineering system using declarative geometry [BoleyHanschke+91/93]; sharable knowledge base on materials engineering/recycling [SintekStein93]/[BoleyBuhrmann+94]; agent-implementation and communication-content language for distributed medical problem solving [CampagnoliLanzola+96].

AVAILABILITY:
All computers and operating systems supporting COMMON LISP (actually, only a subset of CL is needed).
Present version developed on SUN workstations with Lucid CL and CLISP; also runs on Allegro CL, AKCL/GCL.
Interpreter also transformable to C by CLiCC [GoerigkBoley+96].

```
DISTRIBUTION:

RELFUN is currently available freely for research purposes,
preferably via the URL below (see the README under "System" there).
We can also use email for the sources and airmail for the papers and
documentation (the language can be explored just with the interpreter,
minimally ca. 200 kilobytes, and some test examples).
```


```
CONTACT:
Dr. Harold Boley
DFKI
Postfach 2080
D-67608 Kaiserslautern
Germany
Phone: +49-631-205-3459
Fax:   +49-631-205-3210
Email: boley@informatik.uni-kl.de
http://www.dfki.uni-kl.de/~vega/relfun.html
```


1 Getting Started

This section describes the files and the procedure needed to start the **RELFUN** system.

To install RELFUN, first retrieve the file `RELFUN.tar.gz` from the url `http://www.dfki.uni-kl.de/~vega/relfun+/rfm-release/` and unpack it with `gunzip -c RELFUN.tar.gz | tar xf -`, thus creating the directory `RELFUN` with many subdirectories in your current directory (see the directory tree in appendix H).

Now you should switch to the main **RELFUN** directory in which you installed the system. In our environment in Kaiserslautern, you need to change to the directory `/home/rfm/RELFUN` (or at least make a link to it).

In case your LISP system loads the initial file `init.lsp`, you only need to start LISP. Otherwise you have to load **RELFUN** by yourself: `(load "init.lsp")`.

Now the **RELFUN** system will be loaded. If the system was not yet compiled, type `(compile-rfm)`, leave/restart the LISP system (and reload).

Alternatively, you can start the graphical **RELFUN** interface (see figure 1) by changing to the directory `/home/rfm/RELFUN/RFM/tcl` and typing `drl`.

The various parts of the RFM compilation laboratory are:

relfun: the **RELFUN** interpreter

codegenerator: the code generator, which is needed by the compiler

classifier,

normalizer and

mode-interpreter are the 'horizontal' parts of the compiler

gama: the **GWAM**, the WAM emulator which executes the output of the compiler, and the **GAMA**, an abstract machine assembler for the **GWAM**

index: the index-head codegenerator

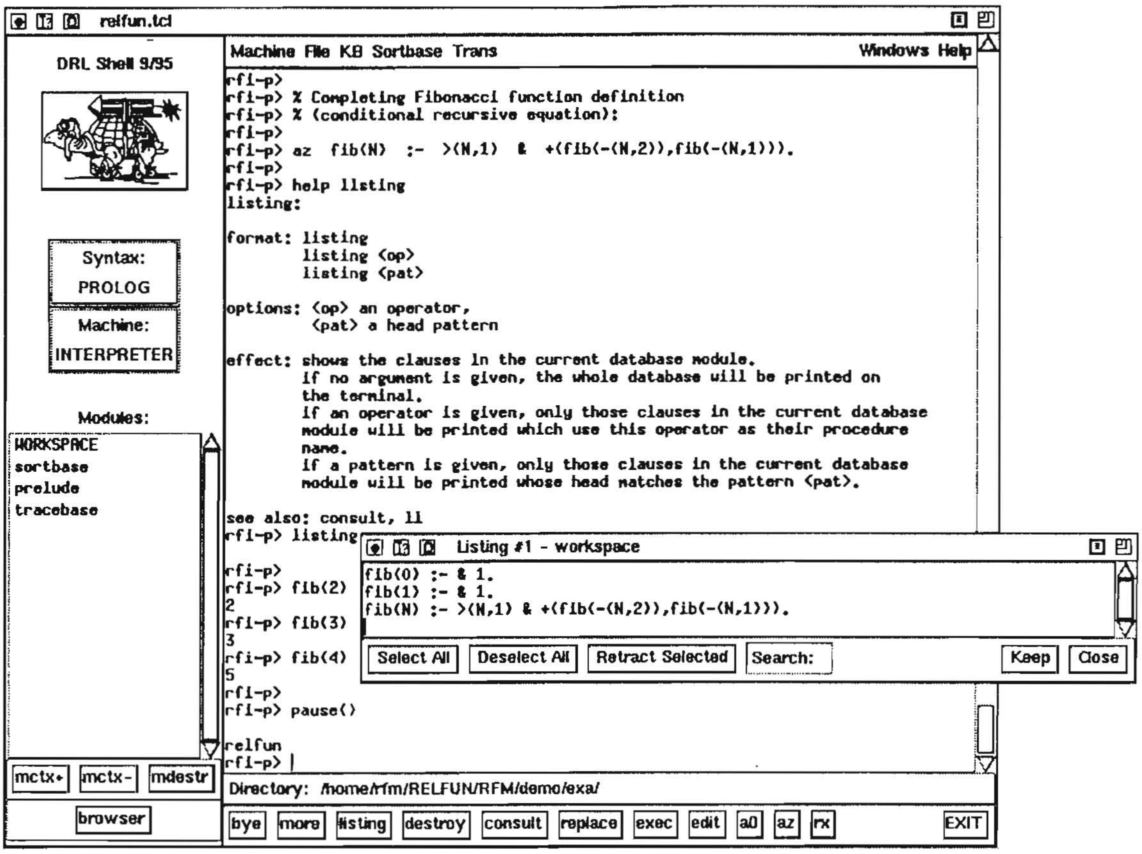

Figure 1: The graphical **RELFUN** interface

Each part of the system resides in its own subdirectory, see appendix H.

Starting RELFUN from LISP:

If all parts of the system are loaded without errors, you should be able to start the **RELFUN** command loop[1] by typing `(relfun)`. To leave the top-level, simply type `bye`.

If you forgot what you can do, type `help` without a keyword to get an overview of the available commands, and `help <command>` to get a detailed description of the command `<command>`. The information you can get with this online help facility is the same collected in this manual within section 6. You can find this documentation in the directory `docu/manual` and the help files in `docu/help`.

Now you are ready to test the system by either

- executing a batch job and watching the automatic dialog scroll on the display,
- consulting a database file and interacting with its clauses, or
- asserting clauses manually and entering commands/goals by hand.

[1]This loop will be further referred to by the name 'top-level'.

To execute a batch job, put a sequence of commands/goals into a batch file `<batchfile>.bat` and type `exec <batchfile>` at the top-level. You can find examples of batch files, including their database files and generated scripts, in the directory `demo` or its subdirectories.

To consult a database, simply type `consult <database>`. This will assert the clauses in the file `<database>` to the current database module. If you do not type an extension, `.rf` or `.rfp` will be used, depending on the current syntax mode (see below).

You can now ask for solutions of goals by typing them into the top-level. If a goal unifies with a clause head the clause will be invoked. If the goal returns a result, you can ask for more solutions by typing `more` or `m` until you get `unknown`. For a short introduction, see the file `brief-intro.rfp` (see section 8), or execute `brief-intro.bat`.

To restart the computation of a previously entered goal, type `ori`. This is especially useful if you have executed `consult` to assert some new clauses and you want to retry a lengthy goal. If you got some experience with the internals of **RELFUN**, you can spy the computed term conjunctions by executing `spy`. This will print the remaining goals instantiated in the current environment. You can control the depth of the conjunction printing with the command `showdepth`. For application programmers the (box-model) `trace` command will normally be more useful.

If the interpreter runs your program without errors, you can compile the database. To execute a goal as a compiled program, you have to switch to the emulator mode with `emul` and compile the database by executing `compile`. The goal should give the same result as in the interpreter mode. The `inter` command will return to the interpreter mode.

In the above description, we mentioned several file name extensions. Here is a complete list of all standard extensions currently used:

`.rf` is the default for loading databases in LISP-style syntax.

`.rfp` is the default for loading databases in PROLOG-style syntax.

`.bat` is the default if you execute a batch file.

`.script` is the choice if you do not specify an extension when generating a script of your session.

`.tex` will be assumed if you type `help` (with a parameter).[2]

These extensions will be used if no other extension is provided. When typing path/file names, you should remember that these are converted to lower case if you do not quote them with double quotes.

2 Builtins and Primitives

This section describes **RELFUN**'s builtins and primitives. They are extended by the definitions in the **RELFUN** prelude (see appendix G). For a tutorial on possible **RELFUN** user definitions, see appendix A.

In the following we will often give constructs in **RELFUN**'s two syntaxes, the PROLOG style followed by the LISP style.

[2]One can simply extend the help facility by adding new files with this default extension in the help directory (`.../RFM/docu/help`). The help files are in a LaTeX format, which become deLaTeXed for online help.

2.1 RELFUN Builtins

The **RELFUN** command `builtins` prints the lists of builtins (functions, predicates, and extras) which can be used in **RELFUN**. The procedural semantics of the arithmetic builtins are the same as in LISP. Also, all other **RELFUN** builtins are based directly on a corresponding standard COMMON LISP function, except the following ones, specially defined as part of **RELFUN**'s COMMON LISP implementation:

- `date()`
  `(date)`
  returns the current date and time

- `operators()`
  `(operators)`
  returns a tuple containing the operators (relations and functions) in the order defined in the database

- `wait(`<*term*>`)`
  `(wait `<*term*>`)`
  prints <*term*> to the output stream and waits for an input (e.g. a newline) from the input stream.

- `break()`
  `(break)`
  enters a COMMON LISP break, which can be left with a LISP dialect-specific command such as `:C` in LUCID.

- `relfun()`
  `(relfun)`
  starts a recursive **RELFUN** shell (described as a "command" in section 6)

2.2 RELFUN Primitives

- `naf(`<*expression*> *)
  `(naf `<*expression*> *)
  not (actually, nand) implementation with negation-as-failure semantics

- `once(`<*expression*> *)
  `(once `<*expression*> *)
  delivers only first solution of <*expression*> *

- `tupof(`<*expression*> *)
  `(tupof `<*expression*> *)
  returns a tuple of all returned terms of <*expression*> *

- `clause(`<*pattern*>`)`
  `(clause `<*pattern*>`)`
  enumerates all clauses matching <*pattern*>

3 The Type System

Types are divided in groups and sorts.

Groups are finite domains or finite exclusions where domains consist of permitted constants and exclusions consist of forbidden (excluded) constants.

Sorts can be builtin or user-defined and have a "$"-prefix. The builtin sorts are "$"-derived from atom subpredicates `atom`, `symbolp`, etc., whereas user-defined sorts are specified as unary predicates in the `sortbase` module.

For more detailed information and examples see appendices B, C and D.

4 Dynamic Signature Unification

A signature clause determines the term pattern (e.g. sorts, cf. section 3) of an operator's arguments and is relevant for those operator clauses which are directly following. So one operator can have several signature clauses, e.g. one for each arity.

The scope of a signature only extends over the current module (cf. section 5), so if the clauses of an operator are to be distributed over several modules (unusual!), one has to create signature clauses for each module.

An example dialog is given in appendix E.

5 The Module System

With the module system it is possible to manage several knowledge bases, each of them containing a sequence of clauses and/or a context, where a context is a sequence of module names. The default module always is `workspace`. Before interpreting a goal, the module extension is created, which is a list of lists of all clauses of the prelude, the current module, and the context modules.

There is an analogy between the module system and the file system, exploited by the rich set of 'm'-commands.

One can find an example dialog in appendix F.

6 Interaction Commands

The following functions and commands, in alphabetical order, are provided by the **RELFUN** interface.

asm:

Short form of **assem**

assem:

Format: `assem [ put <file-name> | get <file-name> |`
`<mod-cmd> [<module-name>] | <list-cmd> [<start> [<end>]] ]`
(where `mod-cmd ::= mod | module | modules` and `list-cmd ::= l | listing`)
Effect: `assem` : assemble compiler database (**compile** and **verti** also
assemble the database, so this command is not yet necessary!),
`assem put <file-name>` : stores the assembler code
of all operators in compiler database in file <file-name>;
compile or **verti** must have been used first! (<file-name> must
be a string enclosed in double quotes),
`assem get <file-name>` : retrieve assembler code from
file <file-name> and link it (this allows to use operators
compiled in previous sessions without having to load the source
code again and compile it; be careful:
listing / **listcode** / **listclass** will not show information
on operators *only* loaded with `assem get`),

`assem modules` (or `mod` / `module` instead of `modules`) :
show interesting information on all current modules
(name, size, position of hashtable, imports),
`assem module <name>` (or `mod` instead of `module`) :
show contents of module <name> (each entry consists of three
cells: label name, address, additional information),
`assem listing [<start> [<end>]]`
(or `l` instead of `listing`) :
list memory section; if <end> is omitted, list 20 cells;
if <start> and <end> are omitted, list next 20 cells
starting one cell behind the last displayed cell).
see also: **horizon, verti, compile**

az:
Format: `az <clause>`
Options: <clause> a **RELFUN** clause
Effect: The <clause> will be inserted at the end of the (possibly empty) current database module.
see also: **a0, azhn, azft, consult, destroy, replace**

azft:
Format: `azft <head> <body1 > ... <bodyn > <foot>`
Options: <head> the head,
<body1 > ... <body$n$ > the bodies,
<foot> the foot of an ft clause.
Effect: The clause `(ft <head> <body1> ... <bodyn> <foot>)` will be inserted at the end of the (possibly empty) current database module. This command is an abbreviation of `az (ft <head> <body1> ... <bodyn> <foot>)`. Not available in P-syntax.
see also: **az, azhn, consult, destroy, replace**

azhn:
Format: `azhn <head> <body1 > ... <bodyn >`
Options: <head> the head,
<body1 > ... <body$n$ > the bodies of an hn clause.
Effect: The clause `(hn <head> <body1> ... <bodyn>)` will be inserted at the end of the (possibly empty) current database module. This command is an abbreviation of `az (hn <head> <body1> ... <bodyn>)`. Not available in P-syntax.
see also: **az, azft, consult, destroy, replace**

a0:
Format: `a0 <clause>`
Options: <clause> a **RELFUN** clause
Effect: The <clause> will be inserted in front of the (possibly empty) current database module.
see also: **az, a0hn, a0ft, consult, destroy, replace**

a0ft:
Format: `a0ft <head> <body1 > ... <bodyn > <foot>`
Options: <head> the head,
<body1 > ... <body$n$ > the bodies, and
<foot> the foot of an ft clause.

Effect: The clause (ft <head> <body1> ... <bodyn> <foot>) will be inserted in front of the (possibly empty) current database module. This command is an abbreviation of a0 (ft <head> <body1> ... <bodyn> <foot>). Not available in P-syntax.
see also: **a0**, **a0hn**, **consult**, **destroy**, **replace**

a0hn:
Format: a0hn <head> <body1 > ... <body$n$ >
Options: <head> the head, and
<body1 > ... <body$n$ > the bodies of an hn clause.
Effect: The clause (hn <head> <body1> ... <bodyn>) will be inserted in front of the (possibly empty) current database module. This command is an abbreviation of a0 (hn <head> <body1> ... <bodyn>). Not available in P-syntax.
see also: **a0**, **a0ft**, **consult**, **destroy**, **replace**

bal2bap:
Format: bal2bap <source-filename> [<dest-filename>]
Options: <source-filename>, <dest-filename>: a pathname (if the pathname contains upper case letters it must be enclosed in double quotes). If no extension is provided, **RELFUN** extends the filenames with ".bat".
Effect: Reads a **RELFUN**-batch, written in L-syntax, from <source-filename>, writes a pretty-printed version in P-syntax to <dest-filename>, resp. standard-output, if no destination is specified.
see also: **rf2rf**, **rf2rfp**, **rfp2rfp**, **rfp2rf**, **bap2bal**

bap2bal:
Format: bap2bal <source-filename> [<dest-filename>]
Options: <source-filename>, <dest-filename>: a pathname (if the pathname contains upper case letters it must be enclosed in double quotes). If no extension is provided, **RELFUN** extends the filenames with ".bat".
Effect: Reads a **RELFUN**-database, written in P-syntax, from <source-filename>, writes a pretty-printed version in L-syntax to <dest-filename>, resp. standard-output, if no destination is specified.
see also: **rf2rf**, **rf2rfp**, **rfp2rfp**, **rfp2rf**, **bal2bap**

break:
Format: break
Effect: For debugging purposes, you can enter the LISP system to inspect the LISP environment by typing the command **break**. The current status of the interpreter will not be changed. Especially you can continue **RELFUN** computing with the command **more** after returning from the LISP system.
Exiting from the LISP system is implementation dependent.
see also: **emul**

browse-sortbase:
Format: browse-sortbase
Effect: If a graphical interface is present, a sortbrowser is started.
see also: **sortbase**

builtins:
Format: builtins
Effect: This command lists the **RELFUN** builtins which can be used in your **RELFUN** programs.

see also: **listing**

bye:
Format: `bye`
Effect: exiting from the current **RELFUN** invocation
see also: **relfun**

classify:
Format: `classify <op>`
Options: <op> an operator
Effect: Produces classified clauses for <op> from the horizontally compiled database. If <op> is omitted, the classified clauses are created for the whole database.
see also: **compile, codegen, verti**

codegen:
Format: `codegen <op>`
Options: <op> an operator
Effect: Produces **GWAM**-code from the classified clauses for <op>. If no operator is given, all classified clauses are transformed.
see also: **compile, classify, verti**

compile:
Format: `compile <op>`
Options: <op> an operator
Effect: If <op> is given, `compile` calls **horizon** to compile the whole **RELFUN** compiler database horizontally and then compiles the operator <op> vertically. If no argument is given, the whole compiler database will be horizontally and vertically compiled.
There is no difference between the command sequence (**horizon, verti**) and the command `compile`.
see also: **consult, horizon, verti**

compile-sortbase:
Format: `compile-sortbase`
Effect: Create an intern structure of the module `sortbase` with direct and indirect subsorts and individuals without changing `sortbase` itself. Cycles are detected.

complete-sortbase:
Format: `complete-sortbase`
Effect: Compare the extensional intersection with the instantiation of the intensional intersection for all pairs of sorts and give an error if there is a discrepancy.
see also: **unique-sortbase**

consult:
Format: `consult <filename>`
Options: <filename> a pathname (if the pathname contains upper case letters it must be enclosed in double quotes)
Effect: Loading a database from file <filename> at the end of the (possibly empty) current database module. If no extension is provided, **RELFUN** extends the filename with `".rf"` or `".rfp"` (depending on the current syntax mode).
see also: **destroy, replace, style**

consult-sortbase:
Format: `consult-sortbase <filename>`

Options: <filename> a pathname (if the pathname contains upper case letters it must be enclosed in double quotes)
Effect: Loading a database from file <filename> at the end of the (possibly empty) module `sortbase`. If no extension is provided, **RELFUN** extends the filename with "`.rf`" or "`.rfp`" (depending on the current syntax mode).
see also: **destroy-sortbase, browse-sortbase, consult**

deanon:
Effect: transform anonymous variables (id) to new (named) variables
see also: **horizon**

destroy:
Format: `destroy`
Effect: Destroy the current database module.
see also: **consult, replace**

destroy-sortbase:
Format: `destroy-sortbase`
Effect: Destroy the entire existing module `sortbase`.
see also: **consult-sortbase**

emul:
Format: `emul` [--nocopy]
Effect: With this command you are entering the emulator mode of **RELFUN** and a flat copy of the current module and its context is generated in compiler database. If --nocopy is given, no copy is generated.
see also: **inter**

endscript:
Format: `endscript`
Effect: If a script has been started, <endscript> will terminate it.
see also: **script**

exec:
Format: `exec <filename>`
Options: <filename> a pathname
Effect: Executing batch commands from the file <filename>. If no extension is provided, **RELFUN** extends the filename with "`.bat`". The batch file may again contain **exec** commands.
see also: **endscript, script**

hash:
Format: `hash` [on|`off`]
Effect: Without argument the current state (on or `off`) of the hash system is displayed.
With argument on hashing is activated.
With argument `off` hashing is deactivated.
Hashing is used to accelerate the interpreter by indexing the clauses of the entire database. Default value is on.

hitrans:
Format: `hitrans`
Effect: higher-order operator calls and higher-order structures are transformed.
See also: **horizon**

horizon:

Format: horizon
Effect: The **RELFUN** database will be horizontally compiled to the **RELFUN** kernel. If you want to compile the database with the command `verti`, use **horizon** first.
see: **compile, verti**

indexing:
Format: `indexing { on | off | :min-clauses <no> | :max-vars <no> | :max-depth <no> | :max-args <no> | :debug on | :debug off }`
Effect: without any option display current settings,
`on (off)` switches indexing on (off),
`:min-clauses <no>` sets the minimal number of clauses for an indexable operator definition to <no>,
`:max-vars <no>` sets the maximal number of variables allowed in a constant block (block-variable-size) to <no>,
`:max-depth <no>` sets the maximal depth of the index tree to <no>,
`:max-args <no>` sets the maximal number of parallelly indexable arguments (index tree breadth) to <no>,
`:debug on (off)`: for internal use only
Mutually excluding options result in executing only the last one.
Example:
`indexing off :min-clauses 5 on :debug off :`
switches indexing on, debugging off and sets min-clauses to 5.

inter:
Format: `inter`
Effect: With this command you are leaving the emulator mode of **RELFUN**, and you return to the interpreter mode.
see also: **emul**

l:
Short form of **listing**

listclass:
Format: `listclass`
`listclass <procedure-name/arity>`
Options: <procedure-name/arity> an operator, including the arity
Effect: Lists the classified version of the procedure. If no argument is specified, the classification of all procedures is listed.
see also: **listing, listcode**

listcode:
Format: `listcode`
`listcode <procedure/arity>`
Options: <procedure/arity> a procedure, including the arity
Effect: Takes the WAM-code and and pretty prints it.
If no argument is specified, the WAM-code of all compiled procedures is listed.
see also: **listing, listclass**

listing:
Format: `listing`
`listing <op>`
`listing <pat>`

Options: <op> an operator,
<pat> a head pattern
Effect: Shows the clauses in the current database module. If no argument is given, the whole database will be printed on the terminal.
If an operator is given, only those clauses in the current database module will be printed which use this operator as their procedure name.
If a pattern is given, only those clauses in the current database module will be printed whose head matches the pattern <pat>.
see also: **consult**, **l**

load:
Format: `load <file1> ...`
Options: <file1>: name of a file (string or symbol)
Effect: Loads files and creates modules. The current module is not changed. If a module already exists in memory it is not reloaded.
see also: **reload**, **msave**, **mhelp**

m:
Short form of **more**

map:
Format: `map <rfi-command> [<module1> ...| --all]`
Options: <rfi-command>: a rfi command, <module1>: name of a module
Effect: Executes the command on the given modules, resp. on all loaded modules.
see also: **mhelp**

mcd:
Format: `mcd [<module>]`
Options: <module>: name of a module (string or symbol)
Effect: Sets the current module to <module>. If the argument is omitted the workspace is taken.
see also: **mhelp**

mconsult:
Format: `mconsult <module1> ...`
Options: <module1>: name of a module (string or symbol)
Effect: Extends the current module by consulting <module1> ..., i.e. it makes copies from the arguments.
see also: mreconsult, mreplace, mhelp

mcreate:
Format: `mcreate <module1> ...`
Options: <module1>: name of a module (string or symbol)
Effect: Creates empty modules in memory. None of the arguments must exist.
see also: **destroy**, **mdestroy**, **mhelp**

mctx:
Format: `mctx [+|-] <module1> ...`
Options: <module1>: name of a module (string or symbol)
Effect: Extends resp. reduces the context of the current module. A context is a list of module names. A goal will be evaluated in the database given by the clauses of the current module and the clauses of the modules in the context. Every module has its own context.
see also: **mctx=**, **mhelp**

mctx=:

Format: `mctx= [<module1>] ...`

Options: <module1>: name of a module (string or symbol)

Effect: Sets the context of the current module. Without argument the context is cleared. A context is a list of module names. A goal will be evaluated in the database given by the clauses of the current module and the clauses of the modules in the context. Every module has its own context.

see also: **mctx, mhelp**

mdestroy:

Format: `mdestroy [<module1> ... | --all]`

Options: <module1>: name of a module (string or symbol)

Effect: Removes modules from memory. With argument `--all` it removes all user modules from memory.

see also: **load, mtell, mhelp**

mflatten:

Format: `mflatten <module1> ...`

Options: <module1>: name of a module (string or symbol)

Effect: Collects the modules and their contexts to create a flat list of clauses. The result is written into the current module.

see also: **emul, mhelp**

mforest:

Format: `mforest <module1> ...`

Options: <module1>: name of a module (string or symbol)

Effect: Shows the module hierarchie in an indented ASCII representation.

see also: **minfo, mhelp**

mhelp:

Format: `mhelp`

Options:

Effect: Lists all commands of the module system with a one-line description.

There is a file RELFUN/RFM/demo/modules/module-demo.bat in the distribution demonstrating the capabilities of the module system by an example.

minfo:

Format: `minfo`

Options:

Effect: Shows context and name of the current module.

see also: **mforest, mhelp**

miser-level:

Format: `miser-level [<level>]`

Options: <level>: an integer.

Effect: Without argument the current miser-level is displayed. With argument the miser-level is set to this value.

The miser-level influences the layout of the pretty printer in P-syntax. There are 4 possible values for <level>:

- `miser-level 0`
  The pretty printer uses many lines and many spaces to produce a readable output.

- `miser-level 1`
  The pretty printer minimizes the use of line breaks, but it still uses many spaces.
- `miser-level 2`
  Like 1, but arguments of structures and functions are separated by commas instead of comma-space-sequences.
- `miser-level 3`
  Like 2, but conjuncts in the body of a clause are separated by commas instead of comma-space-sequences.

mlist:
Format: `mlist [<pattern>]`
Options: <pattern>: a head pattern (see **listing**)
Effect: Searches along the context for the pattern. If the argument is omitted, all accessable clauses are listed.
see also: **listing**, **mhelp**

more:
Format: `more`
Effect: If you asked a goal and you got one solution, you can get the next one. This is only possible if you haven't typed in another goal, and you haven't destroyed (modified) the database.
see also: **m**, **ori**

mreconsult:
Format: `mreconsult <module1> ...`
Options: <module1>: name of a module (string or symbol)
Effect: Makes an update of the current module by reading clauses from a module. Existing predicates are replaced by the new ones. Nonexisting predicates extend the current module.
see also: **reconsult**, **mconsult**, **mreplace**, **mhelp**

mreplace:
Format: `mreplace <module> ...`
Options: <module>: name of a module (string or symbol)
Effect: Destroys the contents of the current module and consults <module>
see also: **mconsult**, **mreconsult**, **mhelp**

msave:
Format: `msave <module1> ...`
Options: <module1>: name of a module (string or symbol)
Effect: Copies the modules from memory to the file system. Includes all sub modules.
see also: **mtell**, **load**, **mhelp**

mtell:
Format: `mtell <module> ...`
Options: <module>: name of a module (string or symbol)
Effect: Writes a copy of the current module to <module>. If <module> exists it is overwritten. If <module> does not exist it is created.
see also: **msave**, **mdestroy**, **mhelp**

nospy:
Format: nospy

Effect: With this **RELFUN** command you leave the trace mode.
see also: **spy**

ori:
Format: `ori`
Effect: The previous goal (not command) will be reasked.
see also: **more**

prelude:
Format: `prelude`
Effect: list contents of prelude
see also: **listing**

print-width:
Format: `print-width [<width>]`
Options: <width>: an integer.
Effect: Without argument the current print-width is displayed. With argument the print-width is set to this value.

reconsult:
Format: `reconsult` <filename>
Options: <filename> a pathname (if the pathname contains upper case letters it must be enclosed in double quotes)
Effect: If no extension is provided, **RELFUN** extends the filename with "`.rf`" or "`.rfp`" (depending on the current syntax mode). **reconsult** makes an update of the current module by reading clauses from a file. Existing predicates are replaced by the new ones. Nonexisting predicates extend the current module.
see also: **consult**, **mreconsult**, **destroy**, **replace**, **style**

relfun:
Format for `style prolog`: `relfun()`
Format within LISP and for `style lisp`: `(relfun)`
Effect: Invoking **RELFUN** from LISP or recursively inside **RELFUN**.
see also: **bye**, **style**

reload:
Format: `reload <file1> ...`
Options: <file1>: name of a file (string or symbol)
Effect: Loads files and replaces existing modules. All modules must exist in memory. The current module is not changed.
see also: **load**, **msave**, **mhelp**

replace:
Format: `replace <filename>`
Options: <filename> a pathname
Effect: Replacing the (possibly empty) current database module with the contents of the file <filename> (for filename syntax and extension, see **consult**)
see also: **destroy**, **consult**

rf2rf:
Format: `rf2rf <source-filename> [<dest-filename>]`
Options: <source-filename>, <dest-filename>: a pathname (if the pathname contains upper case letters it must be enclosed in double quotes). If no extension is provided, **RELFUN** extends the filenames with "`.rf`".

Effect: Reads a **RELFUN**-database, written in L-syntax, from <source-filename>, writes a pretty-printed version to <dest-filename>, resp. standard-output, if no destination is specified.
see also: **rf2rfp**, **rfp2rfp**, **rfp2rf**, **bal2bap**, **bap2bal**

rf2rfp:
Format: `rf2rfp <source-filename> [<dest-filename>]`
Options: <source-filename>, <dest-filename>: a pathname (if the pathname contains upper case letters it must be enclosed in double quotes). If no extension is provided, **RELFUN** extends the source-file with "`.rf`" and the dest-file with "`.rfp`".
Effect: Reads a **RELFUN**-database, written in L-syntax, from <source-filename>, writes a pretty-printed version in P-syntax to <dest-filename>, resp. standard-output, if no destination is specified.
see also: **rf2rf**, **rfp2rfp**, **rfp2rf**, **bal2bap**, **bap2bal**

rfp2rf:
Format: `rfp2rf <source-filename> [<dest-filename>]`
Options: <source-filename>, <dest-filename>: a pathname (if the pathname contains upper case letters it must be enclosed in double quotes). If no extension is provided, **RELFUN** extends the source-file with "`.rfp`" and the dest-file with "`.rf`".
Effect: Reads a **RELFUN**-database, written in P-syntax, from <source-filename>, writes a pretty-printed version in L-syntax to <dest-filename>, resp. standard-output, if no destination is specified.
see also: **rf2rf**, **rf2rfp**, **rfp2rfp**, **bal2bap**, **bap2bal**

rfp2rfp:
Format: `rfp2rfp <source-filename> [<dest-filename>]`
Options: <source-filename>, <dest-filename>: a pathname (if the pathname contains upper case letters it must be enclosed in double quotes). If no extension is provided, **RELFUN** extends the filenames with "`.rfp`".
Effect: Reads a **RELFUN**-database, written in P-syntax, from <source-filename>, writes a pretty-printed version to <dest-filename>, resp. standard-output, if no destination is specified.
see also: **rf2rf**, **rf2rfp**, **rfp2rf**, **bal2bap**, **bap2bal**

rx:
Format: `rx <clause>`
Options: <clause> a **RELFUN** clause
Effect: The <clause> will be removed from the current database module.
see also: **rxft**, **rxhn**, **consult**, **destroy**, **replace**

rxft:
Format: `rxft <head> <body1 > ... <bodyn > <foot>`
Options: <head> the head,
<body1 > ... <body*n* > the bodies, and
<foot> the foot of an ft clause.
Effect: The clause `(ft <head> <body1> ...b <bodyn> <foot>)` will be removed from the current database module. This command is an abbreviation of `rx (ft <head> <body1> ... <bodyn> <foot>)`. Not available in P-syntax.
see also: **rx**, **rxhn**, **consult**, **destroy**, **replace**

rxhn:
Format: `rxhn <head> <body1 > ... <bodyn >`

Options: <head> the head, and
<body1 > ... <bodyn > the bodies of an hn clause.
Effect: The clause `(hn <head> <body1> ... <bodyn>)` will be removed from the current database module. This command is an abbreviation of **rx** `(hn <head> <body1> ... <bodyn>)`. Not available in P-syntax.
see also: **rx, rxft, consult, destroy, replace**

script:
Format: `script <filename>`
Options: <filename> a pathname
Effect: All input and output activity will be logged in the file <filename>. This is useful to analyse traces or to protocol examples stored in a file and run within a batchjob.
see also: **endscript, exec**

showdepth:
Format: `showdepth <n>`
Options: <n> a number
Effect: If the system is in trace mode, the output length of the conjunction printing will be limited to the number <n>. If this number is 0, no limitation exists. This value is the default.
see also: **spy, nospy**

sl:
Short form of `style lisp`

sortbase:
Format: `sortbase`
Effect: Shows the clauses in the module `sortbase`.
see also: **listing, browse-sortbase**

sp:
Short form of `style prolog`

spy:
Format: `spy`
Effect: This activates the spy mode of the **RELFUN** system. After entering the interpreter spy mode, the conjunction of (sub-)goals to be solved is printed on the terminal. The environment is used to instantiate these remaining conjuncts.
see also: **nospy, showdepth**

strict-sortbase:
Format: `strict-sortbase`
Effect: Give an error if a list of `sortbase` individuals is empty.
see also: **complete-sortbase, unique-sortbase**

style:
Format: `style lisp`
`style prolog`
Effect: `style lisp` turns LISP-like syntax on,
`style prolog` turns PROLOG-like syntax on.
See also: **sl, sp**

tell:
Format: `tell <filename>`
Options: <filename> a string or a filename

Effect: The current database module will be stored in the file <filename>. If no extension is provided, **RELFUN** extends the filename with ".rf" or ".rfp".
see also: **consult**, **destroy**, **replace**

timermode:

Format: `timermode [on|off]`
Effect: Without argument the current timermode is displayed.
With argument `on` timermode is activated, i.e. the execution times of goals will be printed.
With argument `off` timermode is deactivated.

trace:

Format: `trace {<procedure>}`

```
where procedure ::= { -all | -rest | {
                      [<head-functor>]
                      [-incl <i-list>]
                      [-excl <i-list>]
                      [-print-p <symbol>] } }
      i-list ::=      {<integer>} | ALL
```

Effect:
Without arguments: Print the names of all traced procedures or print: No procedures traced.
With arguments:

- If the procedure denoted by `<head-functor>` is not traced:
  Activate the tracer on this procedure.
  If `-incl` is specified only the enumerated clauses of the procedure are traced.
  If `-excl` is specified all but the enumerated clauses are traced.
  Else all clauses are traced.
  Clauses are numbered from 1.
  If `-print-p` is given trace information is printed only if this predicate succeeds.
  `-print-p` has the head of the clause as argument.
  At most one of `-incl`, `-excl` can be used.

- If the procedure denoted by `<head-functor>` is already traced:
  Modify the traced clauses.
  If `-incl` is specified the enumerated clauses are traced and the `-print-p`-filter is added if given.
  If `-excl` is specified the enumerated clauses are not traced.
  It is legal to use both `-incl` and `-excl`.

- If trace is called without a `<head-functor>`:
  If `-all` is specified all clauses of all procedures are traced (including those which were explicitly excluded until now).
  If `-rest` is specified all procedures not at all traced yet are traced.
  Besides `-all` or `-rest` no more arguments are considered.

The tracer is based on the box-model of PROLOG. It uses the following symbols at begin of the line to indicate the kind of event:

>	entering a procedure
|	re-entering a procedure
<	exiting a succeeded procedure
~	procedure failed
~!	procedure failed because of running from the right over a cut
~>	procedure failed because there is no head that unifies

A number after the symbol denotes a clause.
After printing ">" and the head of the clause one of "`:-`", "`:- &`" signals a hornish resp. a footed clause.
After printing "<" and the head of the clause "`:-`" indicates a hornish clause, while "`:-&`" followed by the value of the clause indicates a footed clause.
There is a file called `tracer.bat` in the distribution which explains the tracer with examples.
see also: **untrace**

uncomma:
Format: `uncomma`
Effect: remove comma expressions (and inline expandable lambda expressions)
See also: **horizon**

unique-sortbase:
Format: `unique-sortbase`
Effect: Create the intensional intersection for all pairs of sorts and give an error if an intersection contains more than one sort.
see also: **complete-sortbase**

unlambda:
Format: `unlambda`
Effect: remove lambda expressions that cannot be expanded inline
See also: **horizon**

unmacro:
Format: `unmacro`
Effect: remove some macros (let, let*, progn)
See also: **horizon**

unor:
Format: `unor`
Effect: transform or expressions into lambda expressions
See also: **horizon**

untrace
Format: `untrace {<head-functor>}`
Effect:
Without argument: Untrace all traced procedures.
With argument(s): Untrace the given procedure(s).
see also: **trace**

untype:
Format: `untype`
Effect: transform types (dom, exc, and $-terms) as well as bnd- and :-terms
See also: **horizon**

verti:

Format: verti <op>
Options: <op> an operator
Effect: If <op> is given, verti vertically compiles the operator <op>. If no argument is given, the whole compiler database will be vertically compiled.
see also: **compile, horizon, classify, codegen**

7 Access Primitives

Each of the three **RELFUN** (rf) accesses from LISP or another LISP-based system (such as **CoLab**) calls a **RELFUN** primitive whose name corresponds to the first rf argument. Since non-deterministic behavior cannot be directly handled by LISP, the primitives all force the **RELFUN** system to deliver deterministic results:

- (rf 'once <*expression*> *)
  returns the value and bindings for the first solution of the **RELFUN** evaluation of <expression> * (or nil)

- (rf 'naf <*expression*> *)
  is a not implementation with negation-as-failure semantics; returns only t or nil

- (rf 'tupof <*expression*> *)
  returns tuple of returned terms

8 An Introductory Example

We give a small example in PROLOG-style syntax, introducing basic **RELFUN** features. One can find it in the file RFM/demo/sampler/brief-intro.rfp. For further elementary examples see appendix A.

```
%     A BRIEF INTRODUCTION TO BASIC RELFUN, USING ITS PROLOG-LIKE SYNTAX
%                               Harold Boley            Feb 1992 (rev.: July 1996)

% In RELFUN, a LISP list (e1 ... eN) or PROLOG list [e1,...,eN] is equivalent
% to an (N-)tup(le) structure tup[e1,...,eN]. Logical variables are marked by
% a caps- or "_"-initial. The PROLOG list pattern [1,2|V] shortens tup[1,2|V],
% matching instances like tup[1,2,3,4,5] with the binding V = tup[3,4,5].

% A PROLOG clause c(...) :- b1(...), .., bM(...). in RELFUN becomes a kind of
% 'hornish' clause, except that structures, incl. tup-lists, use brackets, not
% parentheses. A conditional equation g(...) = f(...) if b1(...), .., bM(...)
% in RELFUN is generalized to g(...) :- b1(...), .., bM(...) & f(...)., an "&"
% (or 'footed') clause, whose "b" premises may accumulate partial results and
% whose "f" premise returns possibly (non-)ground/(non-)deterministic values.

% As in LISP, nestings like +(*(3,3),1-(8)) => 16 are reduced call-by-value.
% "Passive" structures like +[*[3,3],1-[8]], using brackets, return themselves.
% RELFUN's tup FUNCTION returns the tup STRUCTURE of its evaluated arguments:
% tup(|R) :- & tup[|R].  % R will be bound to the tup of all arguments.
% This permits calls like  T is 3 & tup(*[T,T],1-(8))  =>  tup[*[3,3],7].
```

```
% The below definitions of append and naive reverse illustrate our RELational/
% FUNctional merger (e.g., "is" can invert functions almost as if relations).

% PROLOGish REL style (inversion  revrel(U,tup(1,2))  loops on MORE):

apprel([],L,L).
apprel([H|R],M,[H|S]) :- apprel(R,M,S).
revrel([],[]).
revrel([H|R],L) :- revrel(R,M), apprel(M,[H],L).

% LISP-like FUN style (is-syntax  tup[1,2] is revfun[U]  exhibits inversion):

appfun([],L) :- & L.
appfun([H|R],L) :- & tup(H|appfun(R,L)). %NESTED (user) form ->
% appfun([H|R],L) :- _1 is appfun(R,L) & tup(H|_1).  %FLATTENED
revfun([]) :- & [].
revfun([H|R]) :- & appfun(revfun(R),[H]). %nest and (compiler)
% revfun([H|R]) :- _1 is revfun(R) & appfun(_1,[H]). %flatten

% Using nested or flattened clauses, the RELFUN interpreter allows this dialog:

% rfi-p> apprel(tup(1,2),tup(a),U)     % rfi-p> appfun(tup(1,2),tup(a))
% true                                 % [1,2,a]
% U = [1,2,a]                          %
% rfi-p> apprel(I,J,[1,2,a])           % rfi-p> [1,2,a] is appfun(I,J)
% true                                 % [1,2,a]
% I = []                               % I = []
% J = [1,2,a]                          % J = [1,2,a]
% rfi-p> more    % 4th MORE would fail % rfi-p> more        % 4th MORE would loop
% true                                 % [1,2,a]      % like: for the conjunction
% I = [1]                              % I = [1]            % apprel(I,J, F),
% J = [2,a]                            % J = [2,a]          % [1,2,a] is  F
% rfi-p> revfun(tup(a,b|V))      % A function called with a non-ground "|"-list.
% [b,a]                          % Returned value no. 1 is ground (variableless)
% V = []                         % because V can be bound to the empty list.
% rfi-p> more                    % The request for MORE solutions (PROLOG's ";")
% [H*15,b,a]                     % returns a 3-list pattern starting with H*15,
% V = [H*15]                     % a (free) variable also occurring in V.
% rfi-p> more                    % Another MORE (of infinitely many successes)
% [H*24,H*26,b,a]                % now returns a non-ground list of length 4,
% V = [H*26,H*24]                % whose first two elements reverse those in V.
```

References

[BBK94] Harold Boley, Ulrich Buhrmann, and Christof Kremer. Towards a sharable knowledge base on recyclable plastics. In James K. McDowell and Kenneth J. Meltsner, editors, *Knowledge-Based Applications in Materials Science and Engineering*, pages 29–42. TMS, 1994.

[BEH+96] Harold Boley, Klaus Elsbernd, Hans-Guenther Hein, Thomas Krause, Markus Perling, Michael Sintek, and Werner Stein. RFM Manual: Compiling RELFUN into the Relational/Functional Machine. Document D-91-03, DFKI GmbH, July 1996. Third, Revised Edition.

[BHH+91] Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch, Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, and Frank Steinle. *μCAD2NC: A Declarative Lathe-Workplanning Model Transforming CAD-like Geometries into Abstract NC Programs.* DFKI Document D-91-15, DFKI GmbH, November 1991.

[BHHM95] H. Boley, P. Hanschke, K. Hinkelmann, and M. Meyer. COLAB: A hybrid knowledge representation and compilation laboratory. *Annals of Operations Research*, 55:11–79, 1995. Preprinted as: DFKI Research Report RR-93-08, Jan. 1993.

[Bol90] Harold Boley. A Relational/Functional Language and Its Compilation into the WAM. SEKI Report SR-90-05, Universität Kaiserslautern, Fachbereich Informatik, April 1990.

[Bol92a] Harold Boley. Declarative Operations on Nets. In Fritz Lehmann, editor, *Semantic Networks in Artificial Intelligence*, volume 23, pages 601–637. Special Issue of Computers & Mathematics with Applications, Pergamon Press, 1992.

[Bol92b] Harold Boley. Extended Logic-plus-Functional Programming. In Lars-Henrik Eriksson, Lars Hallnäs, and Peter Schroeder-Heister, editors, *Proceedings of the 2nd International Workshop on Extensions of Logic Programming, ELP '91, Stockholm 1991*, volume 596 of *LNAI*. Springer, 1992.

[Bol93a] Harold Boley. A Direct Semantic Characterization of RELFUN. In Evelina Lamma and Paola Mello, editors, *Proceedings of the 3rd International Workshop on Extensions of Logic Programming, ELP '92, Bologna 1992*, volume 660 of *LNAI*. Springer, 1993.

[Bol93b] Harold Boley. Finite Domains as First-Class Citizens. In Roy Dyckhoff, editor, *Fourth International Workshop on Extensions of Logic Programming, St. Andrews, Scotland, Preprints of the Proceedings*, March 1993.

[GBH+96] Wolfgang Goerigk, Harold Boley, Ulrich Hoffmann, Markus Perling, and Michael Sintek. Komplettkompilation von Lisp: Eine Studie zur Übersetzung von Lisp-Software für C-Umgebungen. *KI*, 2, Juni 1996.

[Hal95] Victoria Hall. Integration von Sorten als ausgezeichnete taxonomische Prädikate in eine relational-funktionale Sprache. Document D-95-04, DFKI GmbH, March 1995.

[Hei89] Hans-Günther Hein. Adding WAM-Instructions to Support Valued Clauses for the Relational/Functional Language RELFUN. SEKI Working Paper SWP-90-02, Universität Kaiserslautern, Fachbereich Informatik, December 1989.

[Kra90] Thomas Krause. Klassifizierte relational/funktionale Klauseln: Eine deklarative Zwischensprache zur Generierung von Register-optimierten WAM-Instruktionen. SEKI Working Paper SWP-90-04, Universität Kaiserslautern, Fachbereich Informatik, May 1990.

[Kra91] Thomas Krause. Globale Datenflußanalyse und horizontale Compilation der relational-funktionalen Sprache RELFUN. Diplomarbeit, DFKI D-91-08, Universität Kaiserslautern, FB Informatik, Postfach 3049, D-6750 Kaiserslautern, March 1991.

[Sin93] Michael Sintek. Indexing PROLOG procedures into DAGs by heuristic classification. DFKI Technical Memo TM-93-05, DFKI GmbH, 1993.

[Sin95] Michael Sintek. FLIP: Functional-plus-logic programming on an integrated platform. Technical Memo TM-95-02, DFKI GmbH, May 1995.

[SSB91] M. Sintek, W. Stein, and U. Buhrmann. Validation and Exploration of the Period System of the Elements: A RELFUN Knowledge Base. In Harold Boley, editor, *A Sampler of Relational/Functional Definitions*, TM-91-04. DFKI GmbH, March 1991. Second, Revised Edition July 1993.

[Ste93] Werner Stein. Indexing Principles for Relational Languages Applied to PROLOG Code Generation. Technical Report Document D-92-22, DFKI GmbH, February 1993.

A Tutorial Dialog

```
rfi-p> exec "conventional.bat"

relfun
rfi-p> %        Developing RELFUN from Conventional Language Constructs
rfi-p> %        =======================================================
rfi-p>
rfi-p> %  Harold Boley       -       Kaiserslautern       -       11-Jul-96
rfi-p>
rfi-p>
rfi-p> help destroy
destroy:

format: destroy

effect: destroy the current database module.

see also: consult, replace
rfi-p> destroy
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Function calls in prefix notation (arithmetical builtins):
rfi-p>
rfi-p> +(3,8)
11
rfi-p> +(3.1415, 8)
11.1415
rfi-p> +(3.1415, 8, 5.0)
16.1415
rfi-p> +(3.1415, *(2,4), 5.0)
16.1415
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Valued conjunctions return values of right-most calls
rfi-p> % (assignments via is-primitive, Variables capitalized):
rfi-p>
rfi-p> Prod is *(2,4), +(3.1415, Prod, 5.0)
```

```
16.1415
Prod=8
rfi-p>
rfi-p> Sum is +(1,2,3,4), Prod is *(1,2,3,4), /(Sum,Prod)
5/12
Sum=10
Prod=24
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Repeated assignments to a single variable must be consistent
rfi-p> % (inconsistent values cause failures signaled by unknown):
rfi-p>
rfi-p> Res is +(1,2,3,4), Res is *(1,2,3,4), /(Res,Res)
unknown
rfi-p>
rfi-p> Res is +(2,2), Res is *(2,2), /(Res,Res)
1
Res=4
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Further builtins in three groups:
rfi-p>
rfi-p> builtins
[functions[+,
           -,
           *,
           /,
           1+,
           1-,
           abs,
           rem,
           floor,
           ceiling,
           truncate,
           round,
           sqrt,
           expt,
           log,
```

```
          sin,
          cos,
          tan,
          asin,
          acos,
          atan,
          max,
          min,
          mod,
          first,
          rest,
          last,
          length,
          intersection,
          union,
          set-difference,
          remove-duplicates,
          gentemp,
          princ-to-string,
          date,
          operators,
          setvar,
          getvar,
          format-to-string,
          format-to-string*,
          code-char,
          instance-classes],
predicates[<,
           <=,
           =,
           /=,
           >,
           >=,
           string<,
           string<=,
           string=,
           string/=,
           string>,
           string>=,
           null,
           atom,
           symbolp,
           numberp,
           integerp,
           plusp,
           minusp],
extras[break,
       readl,
       relfun,
       rf-print,
```

```
        rf-princ,
        rf-terpri,
        rf-fresh-line,
        rf-pprint,
        rf-format,
        pretty-print,
        wait,
        tracer-increment-level,
        tracer-decrement-level,
        tracer-check-max,
        tracer-print-heading,
        tracer-print-head,
        tracer-print-foot,
        tracer-print-hn-or-ft,
        tracer-cps]]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Relation calls (arithmetical predicates):
rfi-p>
rfi-p> numberp(3.1415)
true
rfi-p> numberp(pi)
false
rfi-p>
rfi-p> >(2,1)
true
rfi-p> <=(2,1)
false
rfi-p>
rfi-p> Data is 3.1415, numberp(Data), *(Data,Data)
9.86902225
Data=3.1415
rfi-p> Data is pi, numberp(Data), *(Data,Data)
unknown
rfi-p> % Data is pi,            *(Data,Data)     would be an error!
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Incrementally defined user functions
```

```
rfi-p> % (equating call patterns with returned values):
rfi-p>
rfi-p> help az
az:

format: az <clause>

options: <clause> a relfun clause

effect: the <clause> will be inserted at the end of the
        (possibly empty) current database module.

see also: anull, azhn, azft, consult, destroy, replace
rfi-p>
rfi-p> az  fib(0) :- & 1.
rfi-p> az  fib(1) :- & 1.
rfi-p>
rfi-p> Res is fib(0), Res is fib(1), Res
1
Res=1
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Completing Fibonacci function definition
rfi-p> % (conditional recursive equation):
rfi-p>
rfi-p> az  fib(N)  :-  >(N,1)  &  +(fib(-(N,2)),fib(-(N,1))).
rfi-p>
rfi-p> help listing
listing:

format: listing
        listing <op>
        listing <pat>

options: <op> an operator,
          <pat> a head pattern

effect: shows the clauses in the current database module.
        if no argument is given, the whole database will be printed on
the terminal.
if an operator is given, only those clauses in the current database
        module will be printed which use this operator as their procedure
        name.
if a pattern is given, only those clauses in the current database
```

```
        module will be printed whose head matches the pattern <pat>.

see also: consult, ll
rfi-p> listing
fib(0) :- & 1.
fib(1) :- & 1.
fib(N) :- >(N,1) & +(fib(-(N,2)),fib(-(N,1))).
rfi-p>
rfi-p> fib(2)
2
rfi-p> fib(3)
3
rfi-p> fib(4)
5
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Extensionally defined relations
rfi-p> % (simple facts constitute relational tables):
rfi-p>
rfi-p> az  sister(mary,fred).
rfi-p> az  sister(mary,susan).
rfi-p> az  sister(susan,fred).
rfi-p>
rfi-p> az  father(fred,john).
rfi-p>
rfi-p> help more
more:

format: more

effect: if you asked a goal and you got one solution, you can get
the next one. this is only possible if you haven't typed in
another goal, and you haven't destroyed (modified) the database.

see also: m, ori
rfi-p>
rfi-p> sister(mary,Whose)
true
Whose=fred
rfi-p> more
true
Whose=susan
rfi-p> more
unknown
```

```
rfi-p> father(Who,Whose)
true
Who=fred
Whose=john
rfi-p> more
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Intensional family relationships
rfi-p> % (simple rule defines relational view):
rfi-p>
rfi-p> az  aunt(Female,Person) :- sister(Female,Parent), father(Parent,Person).
rfi-p>
rfi-p> aunt(Who,Whose)
true
Who=mary
Whose=john
rfi-p> more
true
Who=susan
Whose=john
rfi-p> more
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Ground lists:
rfi-p>
rfi-p> 1
1
rfi-p> a
a
rfi-p> [1,a,3,b]
[1,a,3,b]
rfi-p> tup(fib(0),a,fib(3),b)
[1,a,3,b]
rfi-p>
rfi-p> X is [1,a,3,b], rest(X)
[a,3,b]
```

```
X=[1,a,3,b]
rfi-p> X is [1,a,3,b], tup(first(X),rest(X),last(X),length(X))
[1,[a,3,b],[b],4]
X=[1,a,3,b]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Non-ground lists
rfi-p> % (patterns with ordinary as well as rest or "|" variables):
rfi-p>
rfi-p> [X,a,Y,b]
[X,a,Y,b]
rfi-p>
rfi-p> Lst is [X,a,Y,b], Y is 3, Lst
[X,a,3,b]
Lst=[X,a,3,b]
Y=3
rfi-p> Y is 3, Lst is [X,a,Y,b], Lst
[X,a,3,b]
Y=3
Lst=[X,a,3,b]
rfi-p> Lst is [X,a,Y,b], Y is 3, [X,Lst,Y,Lst]
[X,[X,a,3,b],3,[X,a,3,b]]
Lst=[X,a,3,b]
Y=3
rfi-p>
rfi-p> [X,a,Y,b] is [1,a,3,b], [X,Y]
[1,3]
X=1
Y=3
rfi-p> [X,a,X,b] is [1,a,3,b], [X,Y]
unknown
rfi-p> [Head | Tail] is [1,a,3,b], [Head,Tail]
[1,[a,3,b]]
Head=1
Tail=[a,3,b]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Ground and non-ground structures
```

```
rfi-p> % (anonymous or don't-care variables are written as "_"):
rfi-p>
rfi-p> quad[1,a,3,b]
quad[1,a,3,b]
rfi-p> quad[X,a,Y,b]
quad[X,a,Y,b]
rfi-p>
rfi-p> quad[X,a,Y,b] is quad[1,a,3,b]
quad[1,a,3,b]
X=1
Y=3
rfi-p> quad[X,a,X,b] is quad[1,a,3,b]
unknown
rfi-p> quad[X,a,_,b] is quad[1,a,3,b]
quad[1,a,3,b]
X=1
rfi-p> quad[_,a,_,b] is quad[1,a,3,b]
quad[1,a,3,b]
rfi-p> quad[H | T] is quad[X,a,Y,b]
quad[X,a,Y,b]
H=X
T=[a,Y,b]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Calls using rest or "|" variables
rfi-p> % (variables may be "_"-prefixed naturals):
rfi-p>
rfi-p> _1234 is [1,2,3,4], Sum is +(0 | _1234), Prod is *(| _1234)
24
_1234=[1,2,3,4]
Sum=10
Prod=24
rfi-p>
rfi-p>
rfi-p> % Infinitely non-deterministic function
rfi-p> % (recursion can run into the negative):
rfi-p>
rfi-p> az  inftree(N) :- & N.
rfi-p> az  inftree(N) :- & sqrt(N).
rfi-p> az  inftree(N) :- Aux is inftree(-(N,1)) & tree[N,Aux,N].
rfi-p>
rfi-p> pause()

relfun
```

```
rfi-p> bye
true
rfi-p>
rfi-p> inftree(4)
4
rfi-p> more
2.0
rfi-p> more
tree[4,3,4]
rfi-p> more
tree[4,1.7320508075688772,4]
rfi-p> more
tree[4,tree[3,2,3],4]
rfi-p> more
tree[4,tree[3,1.4142135623730952,3],4]
rfi-p> more
tree[4,tree[3,tree[2,1,2],3],4]
rfi-p> more
tree[4,tree[3,tree[2,1.0,2],3],4]
rfi-p> more
tree[4,tree[3,tree[2,tree[1,0,1],2],3],4]
rfi-p> more
tree[4,tree[3,tree[2,tree[1,0.0,1],2],3],4]
rfi-p> more
tree[4,tree[3,tree[2,tree[1,tree[0,-1,0],1],2],3],4]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Finitely non-deterministic function
rfi-p> % (recursion stopped at 0):
rfi-p>
rfi-p> az  fintree(N) :-  >=(N,0) & N.
rfi-p> az  fintree(N) :-  >=(N,0) & sqrt(N).
rfi-p> az  fintree(N) :-  >=(N,0), Aux is fintree(-(N,1))  &  tree[N,Aux,N].
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> fintree(4)
4
rfi-p> more
2.0
```

```
rfi-p> more
tree[4,3,4]
rfi-p> more
tree[4,1.7320508075688772,4]
rfi-p> more
tree[4,tree[3,2,3],4]
rfi-p> more
tree[4,tree[3,1.4142135623730952,3],4]
rfi-p> more
tree[4,tree[3,tree[2,1,2],3],4]
rfi-p> more
tree[4,tree[3,tree[2,1.0,2],3],4]
rfi-p> more
tree[4,tree[3,tree[2,tree[1,0,1],2],3],4]
rfi-p> more
tree[4,tree[3,tree[2,tree[1,0.0,1],2],3],4]
rfi-p> more
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % All-solutions builtin (only for finitely non-deterministic calls!):
rfi-p>
rfi-p> tupof(fintree(4))
[4,
 2.0,
 tree[4,3,4],
 tree[4,1.7320508075688772,4],
 tree[4,tree[3,2,3],4],
 tree[4,tree[3,1.4142135623730952,3],4],
 tree[4,tree[3,tree[2,1,2],3],4],
 tree[4,tree[3,tree[2,1.0,2],3],4],
 tree[4,tree[3,tree[2,tree[1,0,1],2],3],4],
 tree[4,tree[3,tree[2,tree[1,0.0,1],2],3],4]]
rfi-p>
rfi-p> [_,_,_,Fourth|_] is tupof(fintree(4)), Fourth
tree[4,1.7320508075688772,4]
Fourth=tree[4,1.7320508075688772,4]
rfi-p> bye

relfun
```

B Type-System Dialog

```
rfi-p> exec typin

relfun
rfi-p> %                    An Introduction to RELFUN Types
rfi-p> %                    ===============================
rfi-p>
rfi-p> %  Harold Boley      -      Kaiserslautern       -         9-Jul-96
rfi-p>
rfi-p>
rfi-p> % The notion of types in RELFUN encompasses both
rfi-p> % groups and sorts.
rfi-p>
rfi-p> % Groups are finite domains (dom terms),
rfi-p> % specifying permitted constants ('positive' individuals),
rfi-p> % or finite exclusions (exc terms),
rfi-p> % specifying forbidden constants ('negative' individuals).
rfi-p>
rfi-p> % Sorts reuse certain unary predicates as types (with a "$"-prefix).
rfi-p> % Some sorts may be user-defined, employing RELFUN's sortbase;
rfi-p> % other sorts are builtin, i.e. derived from atom subpredicates.
rfi-p>
rfi-p> % Types are first-class citizens, which can be employed anonymously or
rfi-p> % as occurrence bindings of logic variables (as bnd/":"-term rhs's).
rfi-p>
rfi-p> % Orthogonally, types can appear in normal (hn/ft) clauses
rfi-p> % or in signature (sg) clauses.
rfi-p>
rfi-p> destroy
rfi-p> destroy-sortbase
rfi-p>
rfi-p> % Groups
rfi-p> % ------
rfi-p>
rfi-p> % Finite domains
rfi-p> % ..............
rfi-p>
rfi-p> % We might enumerate the cocktails drunk by Mary:
rfi-p>
rfi-p> az drinks(mary,pina-colada).
rfi-p> az drinks(mary,vodka-lemon).
rfi-p> az drinks(mary,orange-flip).
rfi-p>
rfi-p> % This could be queried using 'more' requests,
rfi-p>
rfi-p> drinks(mary,What)
true
What=pina-colada
rfi-p> more
```

```
true
What=vodka-lemon
rfi-p> more
true
What=orange-flip
rfi-p> more
unknown
rfi-p>
rfi-p> % or the 'tupof' primitive:
rfi-p>
rfi-p> tupof(drinks(mary,What),What)
[pina-colada,vodka-lemon,orange-flip]
rfi-p>
rfi-p> destroy
rfi-p>
rfi-p> % Alternatively we may employ a finite domain:
rfi-p>
rfi-p> az drinks(mary,dom[pina-colada,vodka-lemon,orange-flip]).
rfi-p>
rfi-p> % Now a 'closed' solution is obtained by an ordinary query
rfi-p> % (the 'bnd' context around a 'dom' etc. can be ignored here,
rfi-p> % but will be explained in the part on occurrence bindings):
rfi-p>
rfi-p> drinks(mary,What)
true
What=bnd[Anon1*1,dom[pina-colada,vodka-lemon,orange-flip]]
rfi-p>
rfi-p> % Success is obtained by a query with a domain constant:
rfi-p>
rfi-p> drinks(mary,orange-flip)
true
rfi-p>
rfi-p> % Failure is obtained by a query with any non-domain constant:
rfi-p>
rfi-p> drinks(mary,whisky-sour)
unknown
rfi-p>
rfi-p> % A finite domain can also be used in the query,
rfi-p> % where domain unification performs intersection:
rfi-p>
rfi-p> drinks(mary,dom[pina-colada,vodka-lemon,banana-flip])
true
rfi-p>
rfi-p> % We can see dom intersection results by giving them a name:
rfi-p>
rfi-p> drinks(mary,What), dom[pina-colada,vodka-lemon,banana-flip] is What
bnd[_1*0,dom[pina-colada,vodka-lemon]]
What=bnd[_1*0,dom[pina-colada,vodka-lemon]]
rfi-p>
rfi-p> % Another intersection returns a singleton domain, dom[pina-colada],
```

```
rfi-p> % which reduces to its only element, pina-colada:
rfi-p>
rfi-p> drinks(mary,What), dom[pina-colada,banana-flip] is What
pina-colada
What=pina-colada
rfi-p>
rfi-p>
rfi-p> % Finite exclusions
rfi-p> % .................
rfi-p>
rfi-p> % Because we have no negated facts we cannot enumerate
rfi-p> % the cocktails not drunk by John
rfi-p>
rfi-p> % But we may employ a finite exclusion for this purpose:
rfi-p> az drinks(john,exc[whisky-sour,vodka-lemon]).
rfi-p>
rfi-p> % Now a 'negative' solution is obtained by an ordinary query:
rfi-p>
rfi-p> drinks(john,What)
true
What=bnd[Anon8*1,exc[whisky-sour,vodka-lemon]]
rfi-p>
rfi-p> % Success is obtained by a query with any non-excluded constant:
rfi-p>
rfi-p> drinks(john,orange-flip)
true
rfi-p>
rfi-p> % Failure is obtained by a query with an excluded constant:
rfi-p>
rfi-p> drinks(john,whisky-sour)
unknown
rfi-p>
rfi-p> % The unification of a domain and an exclusion subtracts the
rfi-p> % exc elements from the dom elements (failing if none is left):
rfi-p>
rfi-p> drinks(john,dom[pina-colada,orange-flip])
true
rfi-p> drinks(john,What), dom[pina-colada,orange-flip] is What
bnd[_1*0,dom[pina-colada,orange-flip]]
What=bnd[_1*0,dom[pina-colada,orange-flip]]
rfi-p>
rfi-p> drinks(john,dom[pina-colada,whisky-sour])
true
rfi-p> drinks(john,What), dom[pina-colada,whisky-sour] is What
pina-colada
What=pina-colada
rfi-p>
rfi-p> drinks(john,dom[whisky-sour,vodka-lemon])
unknown
rfi-p> drinks(john,What), dom[whisky-sour,vodka-lemon] is What
```

```
unknown
rfi-p>
rfi-p> % What John and Mary can drink together is obtained thus:
rfi-p>
rfi-p> drinks(mary,What), drinks(john,What)
true
What=bnd[Anon27*2,dom[orange-flip,pina-colada]]
rfi-p>
rfi-p> destroy
rfi-p>
rfi-p>
rfi-p> % Sorts
rfi-p> % -----
rfi-p>
rfi-p> % User-defined sorts
rfi-p> % ..................
rfi-p>
rfi-p> % If some 'positive' group of individuals is worth a permanent name
rfi-p> % we can define a unary predicate for it, using facts in the sortbase.
rfi-p> % For example, the cocktails drunk by Mary may be called 'lightmix'.
rfi-p> % A sortbase-defined predicate 'pred' is then usable as the sort '$pred'
rfi-p>
rfi-p> mcd sortbase
Module:  sortbase
Context:
rfi-p>
rfi-p> % A tiny sort lattice, defined by two subsumes 'higher-order' facts:
rfi-p>
rfi-p> az subsumes(cocktail,lightmix).
rfi-p> az subsumes(cocktail,heavymix).
rfi-p>
rfi-p> % Two sort extensions, as facts applying predicates to individuals:
rfi-p>
rfi-p> az lightmix(pina-colada).
rfi-p> az lightmix(vodka-lemon).
rfi-p> az lightmix(orange-flip).
rfi-p>
rfi-p> az heavymix(whisky-sour).
rfi-p> az heavymix("bloody-mary, strong").
rfi-p>
rfi-p> compile-sortbase
rfi-p>
rfi-p> mcd
Module:  workspace
Context:
rfi-p>
rfi-p> % The unification of two such user-defined sorts returns their glb:
rfi-p>
rfi-p> $lightmix is $cocktail
bnd[_2*0,$lightmix]
```

```
rfi-p> $heavymix is $cocktail
bnd[_2*0,$heavymix]
rfi-p> $lightmix is $heavymix
unknown
rfi-p>
rfi-p> % The sort $lightmix abbreviates our domain dom[pina-colada,...]:
rfi-p>
rfi-p> az drinks(mary,$lightmix).
rfi-p>
rfi-p> % Again a 'closed' solution is obtained by the ordinary query
rfi-p>
rfi-p> drinks(mary,What)
true
What=bnd[Anon28*1,$lightmix]
rfi-p>
rfi-p> % success is obtained by a query with a constant in the sort,
rfi-p>
rfi-p> drinks(mary,orange-flip)
true
rfi-p>
rfi-p> % and failure is obtained by a query with any constant not in the sort,
rfi-p>
rfi-p> drinks(mary,whisky-sour)
unknown
rfi-p>
rfi-p> % Again a finite domain can be used in the query,
rfi-p> % where sort-domain unification performs (extensionalized) intersection:
rfi-p>
rfi-p> drinks(mary,dom[pina-colada,vodka-lemon,banana-flip])
true
rfi-p>
rfi-p> % We can see dom intersection results by giving them a name:
rfi-p>
rfi-p> drinks(mary,What), dom[pina-colada,vodka-lemon,banana-flip] is What
bnd[_1*0,dom[pina-colada,vodka-lemon]]
What=bnd[_1*0,dom[pina-colada,vodka-lemon]]
rfi-p>
rfi-p> % A sort can also be used in the query:
rfi-p>
rfi-p> az drinks(fred,vodka-lemon).
rfi-p> drinks(fred,$lightmix)
true
rfi-p>
rfi-p>
rfi-p> % Builtin sorts
rfi-p> % ............
rfi-p>
rfi-p> % The builtin atom subpredicates such as stringp may also
rfi-p> % be used (with a "$"-prefix) as sorts:
rfi-p>
```

```
rfi-p> az drinks(sue,"Barbara's special green-mix").
rfi-p> drinks(sue,$atom)
true
rfi-p> drinks(sue,$numberp)
unknown
rfi-p> drinks(sue,$stringp)
true
rfi-p>
rfi-p> % A builtin sort may be unified with a finite domain,
rfi-p> % performing (generalized) intersection:
rfi-p>
rfi-p> az drinks(jack,dom["Juan's drink",honey-liqueur,"Boston ward 8"]).
rfi-p> drinks(jack,What), $stringp is What
bnd[_1*0,dom["Juan's drink","Boston ward 8"]]
What=bnd[_1*0,dom["Juan's drink","Boston ward 8"]]
rfi-p>
rfi-p> % A builtin sort may also be unified with a user-defined sort:
rfi-p>
rfi-p> $stringp is $heavymix
"bloody-mary, strong"
rfi-p> $symbolp is $heavymix
whisky-sour
rfi-p> $numberp is $heavymix
unknown
rfi-p>
rfi-p>
rfi-p> % Occurrence bindings
rfi-p> % -------------------
rfi-p>
rfi-p> % While the previous types were employed anonymously, they can also be
rfi-p> % associated to logic variables via occurrence bindings (bnd/":"-terms)
rfi-p> % This permits typed variables, forcing group or sort restrictions in a
rfi-p> % rule head, with premises using these variables in arbitrary goals.
rfi-p>
rfi-p> % Domain binding:
rfi-p>
rfi-p> az orders(laura,whisky-sour).
rfi-p>
rfi-p> az drinks(peter,bnd[M,dom[whisky-sour,"bloody-mary, strong"]]) :-
                                                          orders(S,M).
rfi-p> drinks(peter,What)
true
What=whisky-sour
rfi-p>
rfi-p> rx drinks(peter,bnd[M,dom[whisky-sour,"bloody-mary, strong"]]) :-
                                                          orders(S,M).
rfi-p>
rfi-p> % Exclusion binding:
rfi-p>
rfi-p> az drinks(steve,bnd[M,exc[whisky-sour,vodka-lemon]]) :- orders(S,M).
```

```
rfi-p> drinks(steve,What)
unknown
rfi-p>
rfi-p> % User-defined sort binding:
rfi-p>
rfi-p> az drinks(peter,bnd[M,$heavymix]) :- orders(S,M).
rfi-p> drinks(peter,What)
true
What=whisky-sour
rfi-p>
rfi-p> % Builtin sort binding:
rfi-p>
rfi-p> az drinks(adrian,bnd[M,$atom]) :- orders(S,M).
rfi-p> drinks(adrian,What)
true
What=whisky-sour
rfi-p>
rfi-p> % For types (as seen in previous 'What' queries) occurrence bindings are
rfi-p> % also generated internally (by 'deanonymization' through bnd contexts).
rfi-p> % Generally, deanonymization causes correct unification failures for
rfi-p> % inconsistent uses of (list- or structure-)embedded anonymous variables,
rfi-p> % (dom or exc) groups, and (user-defined or builtin) sorts:
rfi-p>
rfi-p> mcd sortbase
Module:  sortbase
Context:
rfi-p>
rfi-p> az person(steve).
rfi-p> az person(john).
rfi-p> az person(mary).
rfi-p>
rfi-p> compile-sortbase
rfi-p> mcd
Module:  workspace
Context:
rfi-p>
rfi-p> spy
rfi-p>
rfi-p> X is [_], [mary] is X
and(X is [_1*0],[mary] is X)
and([mary] is [_1*0])
and([mary])
[mary]
X=[mary]
rfi-p> X is [_], [mary] is X, [john] is X
and(X is [_1*0],[mary] is X,[john] is X)
and([mary] is [_1*0],[john] is [_1*0])
and([john] is [mary])
unknown
rfi-p>
```

```
rfi-p> X is [dom[john,mary]], [mary] is X
and(X is [bnd[_1*0,dom[john,mary]]],[mary] is X)
and([mary] is [bnd[_1*0,dom[john,mary]]])
and([mary])
[mary]
X=[mary]
rfi-p> X is [dom[john,mary]], [mary] is X, [john] is X
and(X is [bnd[_1*0,dom[john,mary]]],[mary] is X,[john] is X)
and([mary] is [bnd[_1*0,dom[john,mary]]],
    [john] is [bnd[_1*0,dom[john,mary]]])
and([john] is [bnd[mary,dom[john,mary]]])
unknown
rfi-p>
rfi-p> X is [exc[fred]], [mary] is X
and(X is [bnd[_1*0,exc[fred]]],[mary] is X)
and([mary] is [bnd[_1*0,exc[fred]]])
and([mary])
[mary]
X=[mary]
rfi-p> X is [exc[fred]], [mary] is X, [john] is X
and(X is [bnd[_1*0,exc[fred]]],[mary] is X,[john] is X)
and([mary] is [bnd[_1*0,exc[fred]]],[john] is [bnd[_1*0,exc[fred]]])
and([john] is [bnd[mary,exc[fred]]])
unknown
rfi-p>
rfi-p> X is [$person], [mary] is X
and(X is [bnd[_1*0,$person]],[mary] is X)
and([mary] is [bnd[_1*0,$person]])
and([mary])
[mary]
X=[mary]
rfi-p> X is [$person], [mary] is X, [john] is X
and(X is [bnd[_1*0,$person]],[mary] is X,[john] is X)
and([mary] is [bnd[_1*0,$person]],[john] is [bnd[_1*0,$person]])
and([john] is [bnd[mary,$person]])
unknown
rfi-p>
rfi-p> X is [$symbolp], [mary] is X
and(X is [bnd[_1*0,$symbolp]],[mary] is X)
and([mary] is [bnd[_1*0,$symbolp]])
and([mary])
[mary]
X=[mary]
rfi-p> X is [$symbolp], [mary] is X, [john] is X
and(X is [bnd[_1*0,$symbolp]],[mary] is X,[john] is X)
and([mary] is [bnd[_1*0,$symbolp]],[john] is [bnd[_1*0,$symbolp]])
and([john] is [bnd[mary,$symbolp]])
unknown
rfi-p>
rfi-p> nospy
```

```
rfi-p>
rfi-p>
rfi-p> % Signatures
rfi-p> % ----------
rfi-p>
rfi-p> % A signature (sg) clause restricts all clauses of a procedure to
rfi-p> % arguments unifying with the sg arguments. Besides constants, variables,
rfi-p> % and structures, the sg arguments can be types.
rfi-p>
rfi-p> % Assuming, everybody drinks soft drinks,
rfi-p>
rfi-p> az drinks(X,soft-drink).
rfi-p>
rfi-p> % we may obtain unexpected results:
rfi-p>
rfi-p> drinks(steve,What)
true
What=soft-drink
rfi-p> drinks(tweety,What)
true
What=soft-drink
rfi-p>
rfi-p> % However, with a drinks signature clause we can prevent this by
rfi-p> % restricting the first argument to be a person:
rfi-p>
rfi-p> style lisp
rfi-l>
rfi-l> a0 (sg (drinks $person _something))
rfi-l>
rfi-l> style prolog
rfi-p>
rfi-p> % Now the bird Tweety no longer drinks soft drinks:
rfi-p>
rfi-p> drinks(steve,What)
true
What=soft-drink
rfi-p> drinks(tweety,What)
unknown
```

C Builtin-Sorts Dialog

```
rfi-p> exec "buisob.bat"

relfun
rfi-p> sp
rfi-p> destroy
rfi-p> inter
rfi-p> sortstyle static
rfi-p> destroy-sortbase
rfi-p>
rfi-p> %  EXAMPLES FOR BUILTIN SORTS
rfi-p> % ============================
rfi-p>
rfi-p>
rfi-p> % Simone Andel
rfi-p>
rfi-p> % The builtin sorts $atom, $symbolp, $stringp, $numberp, $floatp,
rfi-p> % $integerp, $oddp and $evenp are structured as shown below. In contrast
rfi-p> % to user-defined sorts in the sortbase they have infinite extensions.
rfi-p> % This causes an error on unification of exclusions with builtin sorts.
rfi-p> %
rfi-p> %                          $atom
rfi-p> %                         /  |  \
rfi-p> %                        /   |   \
rfi-p> %                 $numberp $symbolp $stringp
rfi-p> %                  /   \
rfi-p> %             $floatp   $integerp
rfi-p> %                         /   \
rfi-p> %                   $evenp    $oddp
rfi-p> %
rfi-p>
rfi-p> % Compute glb of two builtin sorts:
rfi-p> $integerp is $floatp
unknown
rfi-p> $integerp is $oddp
bnd[_2*0,$oddp]
rfi-p> $oddp is $integerp
bnd[_2*0,$oddp]
rfi-p>
rfi-p> % Compute glb of a domain and a builtin sort:
rfi-p> X is dom[1,2,3,4,5,6,a,b], X is $integerp
bnd[X,dom[1,2,3,4,5,6]]
X=dom[1,2,3,4,5,6]
rfi-p> X is dom[1,2,3,4,5,6,a,b], X is $symbolp
bnd[X,dom[a,b]]
X=dom[a,b]
rfi-p> X is dom[1,2,3,4,5,6], X is $oddp
bnd[X,dom[1,3,5]]
X=dom[1,3,5]
```

```
rfi-p>
rfi-p> % Computation of glb of an exclusion and a builtin sort returns an error:
rfi-p> X is exc[1,2,3,4], X is $evenp
error (unify): glb is not computable between exclusion and builtin-sort
rfi-p>
rfi-p> mcd sortbase
Module:  sortbase
Context:
rfi-p> consult "int.rfp"
Reading file "int.rfp"
rfi-p> compile-sortbase
rfi-p> listing
subsumes(int,num).
int(0).
int(1).
int(2).
int(3).
int(4).
int(5).
num(2).
rfi-p> mcd
Module:  workspace
Context:
rfi-p>
rfi-p> % Compute glb of an user-defined and a builtin-sort:
rfi-p> $int is $oddp
bnd[_2*0,dom[1,3,5]]
rfi-p> $int is $integerp
bnd[_2*0,dom[0,1,2,3,4,5]]
rfi-p> $evenp is $int
bnd[_2*0,dom[0,2,4]]
rfi-p> $num is $evenp
2
rfi-p>
rfi-p> % Compute glb of two user-defined sorts:
rfi-p> $int is $num
bnd[_2*0,$num]
rfi-p>
```

D User-Defined Sorts Dialog

```
rfi-p> exec pet.bat

relfun
rfi-p> sp
rfi-p> inter
rfi-p> sortstyle static
rfi-p> destroy
rfi-p> destroy-sortbase
rfi-p>
rfi-p> % Demo of sorts in Interpreter and Compiler
rfi-p> % --------------------------------------------------------------------
rfi-p> % New Relfun commands:  sortbase - for looking at the module sortbase
rfi-p> %                       consult-sortbase - to load the module sortbase
rfi-p> %                       destroy-sortbase - to delete the module sortbase
rfi-p> %                       compile-sortbase - to precompile the module sort
rfi-p> %                       complete-taxonomy - to check if the
rfi-p> %                                           taxonomy is complete,
rfi-p> %                                           i.e. intensional glb
rfi-p> %                                              = extensional glb
rfi-p> %                       unique-glb - to check if the taxonomy is
rfi-p> %                                    unambiguous, i.e. there is at most
rfi-p> %                                    one glb for two sorts.
rfi-p> %                       unsubsumes - subsumes(a,b) -> a(X) :- b(X)
rfi-p> %
rfi-p> %                       sortstyle - to alternate between sort models
rfi-p> %                                   (dynamic and static), or (without
rfi-p> %                                   argument) to indicate the chosen
rfi-p> %                                   model
rfi-p> % compile-sortbase, complete-taxonomy, unique-glb can only be executed
rfi-p> % in the static model.
rfi-p> %
rfi-p> % In the dynamic model it is necessary to make sure that a glb-calculatio
rfi-p> % was loaded in the current database module. Depending on the implementat
rfi-p> % these calculations the "sort knowledge" can be chosen in the subsumes
rfi-p> % notation or Horn logic notation. In this dialog the subsumes notation i
rfi-p> % therefore the command unsubsumes is not needed.
rfi-p>
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> sortstyle
static
rfi-p> % preset static
```

```
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p> %                          STATIC MODEL
rfi-p>
rfi-p> % Loading an incomplete taxonomy
rfi-p> consult-sortbase "exa1"
Reading file "/home/rfm/RELFUN/RFM/demo/types/sorts/exa1.rfp"
rfi-p> %        *a*
rfi-p> %       {1-6}
rfi-p> %     /      \
rfi-p> %    b        g
rfi-p> %  {1-5}  {1,3,6}
rfi-p> %    |        |
rfi-p> %    c        |
rfi-p> %  {1-4}      |
rfi-p> %    |        |
rfi-p> %    d        |
rfi-p> %  {1-3}      |
rfi-p> %    |        |
rfi-p> %   *e*       |
rfi-p> %  {1,2}      |
rfi-p> %       \     |
rfi-p> %        \    f
rfi-p> %            {1}
rfi-p>
rfi-p>
rfi-p> sortbase
subsumes(a,b).
subsumes(b,c).
subsumes(c,d).
subsumes(d,e).
subsumes(e,f).
subsumes(a,g).
subsumes(g,f).
g(6).
g(3).
b(5).
c(4).
d(3).
e(2).
f(1).
rfi-p> pause()

relfun
rfi-p> bye
true
```

```
rfi-p>
rfi-p>
rfi-p> compile-sortbase
rfi-p>
rfi-p> complete-taxonomy
Taxonomy is not complete: B G
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> destroy-sortbase
rfi-p> % Loading an ambiguous taxonomy
rfi-p>
rfi-p> consult-sortbase "exa4"
Reading file "/home/rfm/RELFUN/RFM/demo/types/sorts/exa4.rfp"
rfi-p>
rfi-p>
rfi-p> %  a        b
rfi-p> %  \ \     / /
rfi-p> %   \  \/  /
rfi-p> %    \/  \/
rfi-p> %     c   d
rfi-p> %      \ /
rfi-p> %       e
rfi-p>
rfi-p>
rfi-p>
rfi-p> sortbase
subsumes(a,d).
subsumes(a,c).
subsumes(b,c).
subsumes(b,d).
subsumes(c,e).
subsumes(d,e).
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> compile-sortbase
rfi-p>
rfi-p> complete-taxonomy
rfi-p>
rfi-p>
rfi-p> unique-glb
```

```
Taxonomy is not well defined: A B
rfi-p>
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> destroy-sortbase
rfi-p>
rfi-p> % Loading the "pet sortbase"
rfi-p>
rfi-p> consult-sortbase "pet-base-sub"
Reading file "/home/rfm/RELFUN/RFM/demo/types/sorts/pet-base-sub.rfp"
rfi-p>
rfi-p> compile-sortbase
rfi-p>
rfi-p> sortbase
subsumes(pet,mammal).
subsumes(mammal,dog).
subsumes(mammal,horse).
subsumes(mammal,cat).
subsumes(pet,fish).
subsumes(fish,goldfish).
subsumes(pet,bird).
subsumes(bird,canary).
dog(lassy).
dog(fido).
horse(fury).
cat(tom).
cat(garfield).
goldfish(goldy).
canary(tweety).
rfi-p>
rfi-p> browse-sortbase
rfi-p>
rfi-p> complete-taxonomy
rfi-p>
rfi-p> unique-glb
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> %
rfi-p> % UNIFICATION OF SORTS
rfi-p> %
```

```
rfi-p>
rfi-p> X is $pet, X is $dog
bnd[X,$dog]
X=$dog
rfi-p>
rfi-p> X is $dog, X is $pet
bnd[X,$dog]
X=$dog
rfi-p>
rfi-p> X is $dog, X is $cat
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> X is $dog, X is lassy
lassy
X=lassy
rfi-p>
rfi-p> X is lassy, X is $dog
lassy
X=lassy
rfi-p>
rfi-p> X is $dog, X is fury
unknown
rfi-p>
rfi-p> X is fury, X is $dog
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> X is $dog, X is dom[lassy,fido]
bnd[X,dom[lassy,fido]]
X=dom[lassy,fido]
rfi-p>
rfi-p> X is $dog, X is dom[lassy,fido, fifi]
bnd[X,dom[lassy,fido]]
X=dom[lassy,fido]
rfi-p>
rfi-p> X is $dog, X is dom[lassy,fido, goldy]
bnd[X,dom[lassy,fido]]
X=dom[lassy,fido]
rfi-p>
```

```
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> %
rfi-p> % USE OF SORTS
rfi-p> %
rfi-p>
rfi-p> % Anonymous sorts
rfi-p>
rfi-p> az eats($cat, $fish).
rfi-p>
rfi-p> eats(tom, goldy)
true
rfi-p>
rfi-p>
rfi-p> eats(X, Y)
true
X=bnd[Anon213*1,$cat]
Y=bnd[Anon214*1,$fish]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Named sorts
rfi-p>
rfi-p> az young(tom).
rfi-p> az eats(X : $cat, $bird) :- young(X).
rfi-p>
rfi-p>
rfi-p> l eats
eats($cat,$fish).
eats(X:$cat,$bird) :- young(X).
rfi-p>
rfi-p> eats(X,Y)
true
X=bnd[Anon219*1,$cat]
Y=bnd[Anon220*1,$fish]
rfi-p> m
true
X=tom
Y=bnd[Anon222*1,$bird]
rfi-p> m
```

```
unknown
rfi-p>
rfi-p> eats(tom,tweety)
true
rfi-p>
rfi-p> eats(garfield,tweety)
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> %
rfi-p> % Queries for the "sort base"
rfi-p> %
rfi-p> subsumes(mammal,X)
true
X=dog
rfi-p> m
true
X=horse
rfi-p> m
true
X=cat
rfi-p> m
unknown
rfi-p>
rfi-p> % Listing of the "terminal sorts" immediately possible:
rfi-p> tupof(cat(X),X)
[tom,garfield]
rfi-p>
rfi-p>
rfi-p> % Listing of all the sorts only possible after unsubsumes:
rfi-p> unsubsumes
rfi-p> sortbase
pet(X) :- mammal(X).
mammal(X) :- dog(X).
mammal(X) :- horse(X).
mammal(X) :- cat(X).
pet(X) :- fish(X).
fish(X) :- goldfish(X).
pet(X) :- bird(X).
bird(X) :- canary(X).
dog(lassy).
dog(fido).
horse(fury).
cat(tom).
cat(garfield).
```

```
goldfish(goldy).
canary(tweety).
rfi-p>
rfi-p> tupof(pet(X), X)
[lassy,fido,fury,tom,garfield,goldy,tweety]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> %
rfi-p> % UNARY PREDICATES FOR THE SORT TEST
rfi-p> %
rfi-p> az hunted(X : $bird) :- pet(Y), eats(Y, X).
rfi-p>
rfi-p> hunted(tweety)
true
rfi-p>
rfi-p> hunted(X)
true
X=bnd[Anon371*5,$bird]
rfi-p>
rfi-p>
rfi-p> rx hunted(X : $bird) :- pet(Y), eats(Y, X).
rfi-p>
rfi-p> az hunted(X : $bird) :- eats(Y : $pet, X).
rfi-p>
rfi-p> hunted(tweety)
true
rfi-p>
rfi-p> hunted(X)
true
X=bnd[Anon383*4,$bird]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> %
rfi-p> % In the interpreter mode there is no check to see if a sort has been defined
rfi-p> % in the sort lattice:
rfi-p>
rfi-p> $otto
bnd[_1*0,$otto]
rfi-p>
rfi-p> % In the interpreter mode, since "is" always returns the left-hand side,
```

```
rfi-p> % this has to be expressed by two is-terms (as already seen above).
rfi-p> $pet is $fish
bnd[_2*0,$fish]
rfi-p>
rfi-p> % But
rfi-p> X is $pet, X is $fish
bnd[X,$fish]
X=$fish
rfi-p>
rfi-p> %
rfi-p> % Sorts in compiled RELFUN
rfi-p> %
rfi-p> emul
Collecting modules for the emulator:
sortbase workspace
rfe-p> compile
rfe-p>
rfe-p> $otto
unknown
rfe-p>
rfe-p> $pet is $fish
$fish
rfe-p> $dog is $cat
unknown
rfe-p>
rfe-p> eats(X,Y)
true
X=$cat
Y=$fish
rfe-p> m
true
X=tom
Y=$bird
rfe-p> m
unknown
rfe-p>
rfe-p> $pet is dom[fido,fury,otto]
dom[fido,fury]
rfe-p>
rfe-p> $pet is exc[]
$pet
rfe-p> $pet is exc[goldy]
dom[lassy,fido,fury,tom,garfield,tweety]
rfe-p> $mammal is exc[goldy, fury]
dom[lassy,fido,tom,garfield]
rfe-p> inter
rfi-p> pause()

relfun
rfi-p> bye
```

```
true
rfi-p>
rfi-p>
rfi-p>
rfi-p>
rfi-p> %
rfi-p> %              DYNAMIC MODEL
rfi-p> % Only for use in the inter(preter) mode!
rfi-p>
rfi-p> sortstyle dynamic
rfi-p> sortstyle
dynamic
rfi-p>
rfi-p> destroy
rfi-p> destroy-sortbase
rfi-p>
rfi-p> consult-sortbase "pet-base-sub"
Reading file "/home/rfm/RELFUN/RFM/demo/types/sorts/pet-base-sub.rfp"
rfi-p>
rfi-p> % Loading the glb-calculation ...
rfi-p> consult glb
Reading file "/home/rfm/RELFUN/RFM/demo/types/sorts/glb.rfp"
rfi-p> l
constant-in-sort(Const,Sort) :- Sort(Const) & Const.
constant-in-sort(Const,Sort) :-
        subsumes(Sort,Subsort) & constant-in-sort(Const,Subsort).
greatest-lower-bound(X,Y) :- & remove-subsumed-lbs(all-lbs(X,Y)).
lb(X,X) :- & X.
lb(X,Y) :- subsumes(X,Z) & lb(Z,Y).
lb(X,Y) :- subsumes(Y,Z) & lb(Z,X).
all-lbs(X,Y) :- & remove-duplicates(tupof(lb(X,Y))).
remove-subsumed-lbs(Lbs) :- & rsl(Lbs,[]).
rsl([],Rlbs) :- & Rlbs.
rsl([Lb|Lb-rest],Rlbs) :- &
        rsl(rsl1(Lb,Lb-rest),tup(Lb|rsl1(Lb,Rlbs))).
rsl1(Lb,[]) :- & [].
rsl1(Lb,[Lb1|Rest]) :- subsumes+(Lb,Lb1) !& rsl1(Lb,Rest).
rsl1(Lb,[Lb1|Rest]) :- & tup(Lb1|rsl1(Lb,Rest)).
subsumes+(X,Y) :- subsumes(X,Y).
subsumes+(X,Y) :- subsumes(X,Z), subsumes+(Z,Y).
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> % The following commands do not work in the dynamic model:
rfi-p> compile-sortbase
error - running dynamic sort model
rfi-p> complete-taxonomy
```

```
error - running dynamic sort model
rfi-p> unique-glb
error - running dynamic sort model
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> %
rfi-p> % UNIFICATION OF SORTS
rfi-p> %
rfi-p>
rfi-p> X is $pet, X is $dog
bnd[X,$dog]
X=$dog
rfi-p>
rfi-p> X is $dog, X is $pet
bnd[X,$dog]
X=$dog
rfi-p>
rfi-p> X is $dog, X is $cat
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p> X is $dog, X is lassy
lassy
X=lassy
rfi-p>
rfi-p> X is lassy, X is $dog
lassy
X=lassy
rfi-p>
rfi-p> X is $dog, X is fury
unknown
rfi-p>
rfi-p> X is fury, X is $dog
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
```

```
rfi-p> X is $dog, X is dom[lassy,fido]
bnd[X,dom[lassy,fido]]
X=dom[lassy,fido]
rfi-p>
rfi-p> X is $dog, X is dom[lassy,fido, fifi]
bnd[X,dom[lassy,fido]]
X=dom[lassy,fido]
rfi-p>
rfi-p> X is $dog, X is dom[lassy,fido, goldy]
bnd[X,dom[lassy,fido]]
X=dom[lassy,fido]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> %
rfi-p> % USE OF SORTS
rfi-p> %
rfi-p>
rfi-p> % Anonymous sorts
rfi-p>
rfi-p> az eats($cat, $fish).
rfi-p>
rfi-p> eats(tom, goldy)
true
rfi-p>
rfi-p>
rfi-p> eats(X, Y)
true
X=bnd[Anon702*1,$cat]
Y=bnd[Anon703*1,$fish]
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> % Named sorts
rfi-p>
rfi-p> az young(tom).
rfi-p> az eats(X : $cat, $bird) :- young(X).
rfi-p>
rfi-p>
rfi-p> eats(X,Y)
```

```
true
X=bnd[Anon704*1,$cat]
Y=bnd[Anon705*1,$fish]
rfi-p> m
true
X=tom
Y=bnd[Anon707*1,$bird]
rfi-p> m
unknown
rfi-p>
rfi-p> eats(tom,tweety)
true
rfi-p>
rfi-p> eats(garfield,tweety)
unknown
rfi-p>
rfi-p> pause()

relfun
rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p>
rfi-p> %
rfi-p> % Queries for the "sort base"
rfi-p> %
rfi-p> subsumes(mammal,X)
true
X=dog
rfi-p> m
true
X=horse
rfi-p> m
true
X=cat
rfi-p> m
unknown
```

E Dynamic-Signatures Dialog

```
rfi-p> exec "dyn-sg.bat"

relfun
rfi-p> sp
rfi-p> destroy
rfi-p> inter
rfi-p> sortstyle static
rfi-p> destroy-sortbase
rfi-p> sl
rfi-l> ;  EXAMPLE DIALOG ON DYNAMIC SIGNATURE UNIFICATION
rfi-l> ; =================================================
rfi-l>
rfi-l> ; Simone Andel
rfi-l>
rfi-l> ; A signature clause which determines the sort or type of an argument
rfi-l> ; always has to be placed before the procedure clauses it belongs to.
rfi-l> ; Its scope extends over the following clauses until the next
rfi-l> ; signature clause.
rfi-l>
rfi-l> consult "bsp.rf"
Reading file "bsp.rf"
rfi-l> l
(sg (p 5 _z))
(ft (p _x _y)
    (+ _x _y) )
rfi-l> (p 5 3)
8
rfi-l> (p 6 3)
unknown
rfi-l> ; the number 5 is requested by the signature clause for p
rfi-l> (p _z 3)
8
(_z = 5)
rfi-l>
rfi-l> destroy
rfi-l> ; signature clause for empty procedure
rfi-l> az (sg (p 5 _x))
rfi-l> listing
(sg (p 5 _x))
rfi-l> (p 5 3)
unknown
rfi-l>
rfi-l> sp
rfi-p> destroy
rfi-p> mcd sortbase
Module:  sortbase
Context:
rfi-p> consult "veb-base-sub.rfp"
```

```
Reading file "veb-base-sub.rfp"
rfi-p> compile-sortbase
rfi-p> listing
dog(lassy).
dog(fido).
cat(tom).
cat(garfield).
goldfish(goldy).
canary(tweety).
rfi-p>
rfi-p> mcd
Module:  workspace
Context:
rfi-p> consult "bsp1.rf"
Reading file "bsp1.rf"
rfi-p> listing
age(tom) :- & 4.
age(garfield) :- & 6.
age(goldy) :- & 1.
sg(young($fish))
young(X) :- <(age(X),2).
sg(young($cat))
young(X) :- <(age(X),5).
sg(eats($cat,$fish))
eats(X,Y) :- young(X).
rfi-p> % Dynamic signature unification for more than one signature clause
rfi-p> % in the procedure
rfi-p>
rfi-p> young(goldy)
true
rfi-p> m
unknown
rfi-p> young(tom)
true
rfi-p> young(X)
true
X=goldy
rfi-p> m
true
X=tom
rfi-p> m
unknown
rfi-p> eats(tom, goldy)
true
rfi-p> eats(tom, fido)
unknown
rfi-p> eats(garfield, goldy)
unknown
rfi-p>
rfi-p> destroy
```

```
rfi-p> consult "rev.rf"
Reading file "rev.rf"
rfi-p> listing
sg(rev(tup(|_),tup(|_)))
rev(L,R) :- rev(L,[],R).
sg(rev(tup(|_),tup(|_),tup(|_)))
rev([],L,L).
rev([Y|L1],L2,R) :- rev(L1,[Y|L2],R).
rfi-p> rev([1,2,3],A)
true
A=[3,2,1]
rfi-p> m
unknown
rfi-p> rev(L,[3,2,1])
true
L=[1,2,3]
rfi-p> rev([1,2,3],[],L)
true
L=[3,2,1]
rfi-p> rev([1,2,3]|A)
true
A=[[3,2,1]]
rfi-p> m
true
A=[[|Anon736*2],[3,2,1|Anon736*2]]
rfi-p> m
unknown
rfi-p> rev([],1,1)
unknown
rfi-p> % 1 is not a list
rfi-p>
rfi-p> sl
rfi-l> rx (sg (rev (tup | id) (tup | id)))
rfi-l> rx (sg (rev (tup | id) (tup | id) (tup | id)))
rfi-l> sp
rfi-p> listing
rev(L,R) :- rev(L,[],R).
rev([],L,L).
rev([Y|L1],L2,R) :- rev(L1,[Y|L2],R).
rfi-p> rev([],1,1)
true
rfi-p> % without signature: true although 1 is not a list
rfi-p>
rfi-p> destroy
rfi-p> destroy-sortbase
rfi-p> mcd sortbase
Module:  sortbase
Context:
rfi-p> consult "int-and-real.rfp"
Reading file "int-and-real.rfp"
```

```
rfi-p> compile-sortbase
rfi-p> listing
woman(peggy_bundy).
woman(marcy_darcy).
man(al_bundy).
man(jefferson_darcy).
boy(bud_bundy).
girl(kelly_bundy).
dog(buck).
fish(marcies_goldfish).
int(0).
int(1).
int(2).
int(3).
int(4).
int(5).
rfi-p> mcd
Module:  workspace
Context:
rfi-p>
rfi-p> consult "fib-sg.rf"
Reading file "fib-sg.rf"
rfi-p> listing
sg(fib($int))
fib(0) :- & 1.
fib(1) :- & 1.
fib(N) :- & +(fib(-(N,1)),fib(-(N,2))).
sg(fib(null))
fib(null) :- add1(null,One) & One.
sg(fib(s(X)))
fib(One) :- add1(null,One) & One.
fib(N) :-
         sub1(N,Nm1),
         sub1(Nm1,Nm2),
         R1 is fib(Nm1),
         R2 is fib(Nm2),
         plus(R1,R2,R) &
         R.
sub1(s[N],N).
add1(N,s[N]).
plus(null,Y,Y).
plus(X,Y,R) :- sub1(X,Xm1), add1(Y,Yp1), plus(Xm1,Yp1,R).
rfi-p> fib(null)
s[null]
rfi-p> fib(s[null])
s[null]
rfi-p> fib(s[s[s[s[s[null]]]]])
s[s[s[s[s[s[s[s[null]]]]]]]]
rfi-p> fib(0)
1
```

```
rfi-p> fib(1)
1
rfi-p> fib(5)
8
rfi-p> fib(11)
unknown
rfi-p>
rfi-p> destroy
rfi-p> sl
rfi-l> az (sg (likes $woman $man))
rfi-l> az (hn (likes peggy_bundy al_bundy))
rfi-l> az (sg (hates $woman $man))
rfi-l> az (hn (hates peggy_bundy al_bundy))
rfi-l> sp
rfi-p> listing
sg(likes($woman,$man))
likes(peggy_bundy,al_bundy).
sg(hates($woman,$man))
hates(peggy_bundy,al_bundy).
rfi-p> P(peggy_bundy,al_bundy)
[peggy_bundy,al_bundy]
P=tup
rfi-p> m
true
P=likes
rfi-p> m
true
P=hates
rfi-p>
rfi-p> destroy
rfi-p> sl
rfi-l> az (ft (plus _x _y) (+ _x _y))
rfi-l> listing
(ft (plus _x _y)
    (+ _x _y) )
rfi-l> (plus 5 0)
5
rfi-l> (plus 5 peggy_bundy)
error - the value of number2, peggy_bundy, should be a number
rfi-l> ; gives an error
rfi-l>
rfi-l> a0 (sg (plus $int $int))
rfi-l> listing
(sg (plus $int $int))
(ft (plus _x _y)
    (+ _x _y) )
rfi-l> (plus 5 0)
5
rfi-l> (plus 5 peggy_bundy)
unknown
```

```
rfi-l> ; fails at signature clause
rfi-l>
rfi-l> destroy
rfi-l> destroy-sortbase
rfi-l> consult "qsort.rf"
Reading file "qsort.rf"
rfi-l> sp
rfi-p> listing
sg(qsort(Cr)(tup()))
qsort[Cr]([]) :- & [].
sg(qsort(Cr)(tup(_|_)))
qsort[Cr]([X|Y]) :-
         partition[Cr](X,Y,Sm,Gr) &
         appfun(qsort[Cr](Sm),tup(X|qsort[Cr](Gr))).
partition[Cr](X,[Y|Z],[Y|Sm],Gr) :-
         Cr(Y,X), partition[Cr](X,Z,Sm,Gr).
partition[Cr](X,[Y|Z],Sm,[Y|Gr]) :-
         Cr(X,Y), partition[Cr](X,Z,Sm,Gr).
partition[Cr](X,[X|Z],Sm,Gr) :- partition[Cr](X,Z,Sm,Gr).
partition[Cr](X,[],[],[]).
appfun([],L) :- & L.
appfun([H|R],L) :- & tup(H|appfun(R,L)).
second<([_,N],[_,M]) :- <(N,M).
first<(found[N,_],found[M,_]) :- string<(N,M).
rfi-p>
rfi-p> % Higher-order procedures with signature clauses
rfi-p> qsort[<]([3,1,2])
[1,2,3]
rfi-p> qsort[second<]([[ma,2],[lu,3],[kl,1]])
[[kl,1],[ma,2],[lu,3]]
rfi-p> qsort[<](f[3,2,1])
unknown
```

F Module-System Dialog

```
rfi-p> exec module-demo

rfi-p>
rfi-p> %\title{RelFun's Module System}
rfi-p>
rfi-p>
rfi-p> % module-demo.bat:  Demonstration of the module system
rfi-p> %                   by a simple arithmetic example.
rfi-p> %
rfi-p> %
rfi-p> % Michael Herfert, Feb/Aug 1995
rfi-p>
rfi-p>
rfi-p> % This dialog is divided into six sections:
rfi-p>
rfi-p> %       1. A first look at the example modules
rfi-p> %       2. Contexts and the current module
rfi-p> %       3. Analogy between modules and files
rfi-p> %       4. File related commands of the module system
rfi-p> %       5. Short review of all module commands
rfi-p> %       6. Modules and the emulator
rfi-p> %       Appendix
rfi-p>
rfi-p>
rfi-p> %\section{1. A first look at the example modules}
rfi-p>
rfi-p>
rfi-p> % Consider a system of modules having the following structure:
rfi-p> %
rfi-p> %
rfi-p> %                             facfib
rfi-p> %                               |
rfi-p> %                               |
rfi-p> %                               |
rfi-p> %                           arithmetic
rfi-p> %                               |
rfi-p> %                               |
rfi-p> %                               |
rfi-p> %                 |-------------------------------------|
rfi-p> %                 |                                     |
rfi-p> %                 |                                     |
rfi-p> %            predsucc-num                         predsucc-sym
rfi-p>
rfi-p>
rfi-p> % predsucc-num  is a module doing primitive operations
rfi-p> %               (predecessor and successor) on numbers
rfi-p> %               given in numeric representation.
rfi-p>
```

```
rfi-p> % predsucc-sym  is a module doing the same primitive operations
rfi-p> %               on numbers given in symbolic representation.
rfi-p> %               In symbolic representation numbers are RelFun-structure
rfi-p> %               zero is represented as 0, one as s[0], two as s[s[0]] e
rfi-p>
rfi-p> % arithmetic    is a module implementing 'higher' arithmetic operations
rfi-p> %               like addition and multiplication. It is independent of
rfi-p> %               representation. It uses the primitive operations given
rfi-p> %               by predsucc-num resp. predsucc-sym.
rfi-p>
rfi-p> % facfib        is a module implementing the well known faculty and
rfi-p> %               fibonacci operations.
rfi-p>
rfi-p>
rfi-p> % With RelFun's module system it is possible to select dynamically modu
rfi-p> % for the goal evaluator.
rfi-p> % For example one can select the three modules predsucc-num, arithmetic
rfi-p> % facfib to do arithmetic operations on numbers given in the usual way.
rfi-p> % Or one can select the three modules predsucc-sym, arithmetic and
rfi-p> % facfib to do arithmetic operations on numbers given as structures.
rfi-p>
rfi-p>
rfi-p> inter                % switch on interpreter mode
rfi-p> destroy              % make a clean memory
rfi-p> mdestroy --all
All user modules have been destroyed.
rfi-p>
rfi-p> % Create a module named predsucc-num
rfi-p> % and consult the file predsucc-num.rfp into it:
rfi-p> load predsucc-num
Reading file "predsucc-num.rfp"
Creating module predsucc-num
rfi-p>
rfi-p> % 'mforest' shows the modules and their hierarchie.
rfi-p> % workspace is the default module (like user-package in lisp),
rfi-p> % prelude, sortbase and tracebase are system modules.
rfi-p> mforest
workspace
sortbase
prelude
tracebase
predsucc-num
rfi-p> pause()
true
rfi-p>
rfi-p> % load the other modules:
rfi-p> load predsucc-sym arithmetic facfib
Reading file "predsucc-sym.rfp"
Creating module predsucc-sym
Reading file "arithmetic.rfp"
```

```
Creating module arithmetic
Reading file "facfib.rfp"
Creating module facfib
rfi-p> mforest
workspace
sortbase
prelude
tracebase
predsucc-num
predsucc-sym
arithmetic
facfib
rfi-p> pause()
true
rfi-p>
rfi-p> % The command 'map' executes a command on the modules given as arguments.
rfi-p> % Let's look at the simple source code of the examples.
rfi-p> % Note that the clauses of predsucc-num and predsucc-sym have the same name.
rfi-p> map listing predsucc-num predsucc-sym arithmetic facfib
---- Doing listing on module predsucc-num:
sub1(N,R) :- R is 1-(N).
add1(N,R) :- R is 1+(N).
---- Doing listing on module predsucc-sym:
sub1(s[N],N).
add1(N,s[N]).
---- Doing listing on module arithmetic:
plus(0,Y,Y).
plus(X,Y,R) :- sub1(X,Xm1), add1(Y,Yp1), plus(Xm1,Yp1,R).
mult(0,Y,0).
mult(X,Y,R) :- sub1(X,Xm1), mult(Xm1,Y,Rmy), plus(Y,Rmy,R).
---- Doing listing on module facfib:
fac(0,R) :- add1(0,R).
fac(N,R) :- sub1(N,Nm1), fac(Nm1,S), mult(N,S,R).
fib(0,One) :- add1(0,One).
fib(One,One) :- add1(0,One).
fib(N,R) :-
        sub1(N,Nm1),
        sub1(Nm1,Nm2),
        fib(Nm1,R1),
        fib(Nm2,R2),
        plus(R1,R2,R).
rfi-p> pause()
true
rfi-p>
rfi-p>
rfi-p> %\section{2. Contexts and the current module}
rfi-p>
rfi-p>
rfi-p> % A module is a collection of data in memory.
rfi-p> % It is made of clauses and a context register.
```

```
rfi-p> % The context register contains a list of module names.
rfi-p> % When evaluating a goal RelFun sees all clauses of the current module
rfi-p> % and all clauses of the modules in the context.
rfi-p> % Every module can be made the current module with the command 'mcd'.
rfi-p> % By default the current module is workspace.
rfi-p>
rfi-p> % The following goal yields 'unknown' because their are two modules
rfi-p> % (predsucc-num and predsucc-sym) in memory defining clauses for 'add1'
rfi-p> % but none is activated:
rfi-p> add1(0, X)
unknown
rfi-p>
rfi-p> % The current module (workspace) is empty so this gives no output:
rfi-p> listing
rfi-p> pause()
true
rfi-p>
rfi-p> % Now activate predsucc-num by setting the context:
rfi-p> mctx= predsucc-num
Context: predsucc-num
rfi-p>
rfi-p> % 'mlisting' lists the current module and the modules of its context:
rfi-p> mlisting
---- workspace:
---- predsucc-num:
sub1(N,R) :- R is 1-(N).
add1(N,R) :- R is 1+(N).
rfi-p> add1(0, X)
true
X=1
rfi-p>
rfi-p> % 'plus' is not yet activated:
rfi-p> plus(3,4,X)
unknown
rfi-p> mctx + arithmetic                        % extend the context
Context: predsucc-num arithmetic
rfi-p> plus(3,4,X)
true
X=7
rfi-p> pause()
true
rfi-p>
rfi-p> % Activate the full package:
rfi-p> mctx + facfib
Context: predsucc-num arithmetic facfib
rfi-p>
rfi-p> % In general the module hierarchie is a DAG.
rfi-p> % The ascii representation via 'mforest' prints a forest,
rfi-p> % but even if some modules occur more then one time,
rfi-p> % they are unique in memory.
```

```
rfi-p> mforest
workspace
  predsucc-num
  arithmetic
  facfib
sortbase
prelude
tracebase
predsucc-num
predsucc-sym
arithmetic
facfib
rfi-p> mlisting
---- workspace:
---- predsucc-num:
sub1(N,R) :- R is 1-(N).
add1(N,R) :- R is 1+(N).
---- arithmetic:
plus(0,Y,Y).
plus(X,Y,R) :- sub1(X,Xm1), add1(Y,Yp1), plus(Xm1,Yp1,R).
mult(0,Y,0).
mult(X,Y,R) :- sub1(X,Xm1), mult(Xm1,Y,Rmy), plus(Y,Rmy,R).
---- facfib:
fac(0,R) :- add1(0,R).
fac(N,R) :- sub1(N,Nm1), fac(Nm1,S), mult(N,S,R).
fib(0,One) :- add1(0,One).
fib(One,One) :- add1(0,One).
fib(N,R) :-
        sub1(N,Nm1),
        sub1(Nm1,Nm2),
        fib(Nm1,R1),
        fib(Nm2,R2),
        plus(R1,R2,R).
rfi-p> fac(3,X)
true
X=6
rfi-p> pause()
true
rfi-p>
rfi-p> % Now we use the symbolic representation of numbers:
rfi-p> mctx - predsucc-num + predsucc-sym
Context: arithmetic facfib predsucc-sym
rfi-p> add1(0, X)
true
X=s[0]
rfi-p> fac(s[s[s[0]]], X)
true
X=s[s[s[s[s[s[0]]]]]]
rfi-p> mlisting
---- workspace:
```

```
---- arithmetic:
plus(0,Y,Y).
plus(X,Y,R) :- sub1(X,Xm1), add1(Y,Yp1), plus(Xm1,Yp1,R).
mult(0,Y,0).
mult(X,Y,R) :- sub1(X,Xm1), mult(Xm1,Y,Rmy), plus(Y,Rmy,R).
---- facfib:
fac(0,R) :- add1(0,R).
fac(N,R) :- sub1(N,Nm1), fac(Nm1,S), mult(N,S,R).
fib(0,One) :- add1(0,One).
fib(One,One) :- add1(0,One).
fib(N,R) :-
         sub1(N,Nm1),
         sub1(Nm1,Nm2),
         fib(Nm1,R1),
         fib(Nm2,R2),
         plus(R1,R2,R).
---- predsucc-sym:
sub1(s[N],N).
add1(N,s[N]).
rfi-p> pause()
true
rfi-p>
rfi-p> % It is possible to backtrack across modules:
rfi-p> mctx + predsucc-num
Context: arithmetic facfib predsucc-sym predsucc-num
rfi-p> add1(0, X)
true
X=s[0]
rfi-p> more
true
X=1
rfi-p> more
unknown
rfi-p> pause()
true
rfi-p> mctx - predsucc-num
Context: arithmetic facfib predsucc-sym
rfi-p>
rfi-p> % Make facfib the current module:
rfi-p> mcd facfib
Module:  facfib
Context:
rfi-p>
rfi-p> % Now every rfi command works on this module:
rfi-p> listing
fac(0,R) :- add1(0,R).
fac(N,R) :- sub1(N,Nm1), fac(Nm1,S), mult(N,S,R).
fib(0,One) :- add1(0,One).
fib(One,One) :- add1(0,One).
fib(N,R) :-
```

```
        sub1(N,Nm1),
        sub1(Nm1,Nm2),
        fib(Nm1,R1),
        fib(Nm2,R2),
        plus(R1,R2,R).
rfi-p> pause()
true
rfi-p>
rfi-p>
rfi-p> %\section{3. Analogy between modules and files}
rfi-p>
rfi-p>
rfi-p> % One base of RelFun's module system is the correspondence to the filesystem.
rfi-p> % For example there are the analogies:
rfi-p> %
rfi-p> %   a file         is a collection of data on disk
rfi-p> %   a module       is a collection of data in memory
rfi-p> %
rfi-p> %   'consult'      reads clauses from a file to extend the current module
rfi-p> %   'mconsult'     reads clauses from a module to extend the current module
rfi-p> %
rfi-p> %   'tell'         writes the clauses of the current module to a file.
rfi-p> %   'mtell'        writes the clauses of the current module to a module
rfi-p> %
rfi-p> %   In a similiar way there are analogies between
rfi-p> %   'replace'/'mreplace', 'reconsult'/'mreconsult'.
rfi-p> %
rfi-p> %   In Unix one filename can be used in multiple branches of the directory
rfi-p> %   tree to name different files.
rfi-p> %   The module system in opposite demands uniq module names.
rfi-p> %   This makes references to modules non-ambiguous from every position in
rfi-p> %   the DAG.
rfi-p>
rfi-p>
rfi-p> % Make workspace the current module:
rfi-p> mcd
Module:  workspace
Context: arithmetic facfib predsucc-sym
rfi-p> % 'mconsult' consults clauses from the module given as argument by copying
rfi-p> % them into the current module:
rfi-p> mconsult facfib
Warning:  facfib is still in the current context.
rfi-p> listing
fac(0,R) :- add1(0,R).
fac(N,R) :- sub1(N,Nm1), fac(Nm1,S), mult(N,S,R).
fib(0,One) :- add1(0,One).
fib(One,One) :- add1(0,One).
fib(N,R) :-
        sub1(N,Nm1),
        sub1(Nm1,Nm2),
```

```
        fib(Nm1,R1),
        fib(Nm2,R2),
        plus(R1,R2,R).
rfi-p> pause()
true
rfi-p>
rfi-p> % Now we remove facfib from the context of workspace:
rfi-p> mctx - facfib
Context: arithmetic predsucc-sym
rfi-p>
rfi-p> % The following goal succeeds because workspace has a copy of facfib:
rfi-p> fac(s[s[s[0]]], X)
true
X=s[s[s[s[s[s[0]]]]]]
rfi-p> pause()
true
rfi-p>
rfi-p> % Remove clauses from the workspace:
rfi-p> rx fac(0, R) :- add1(0, R).
rfi-p> rx fac(N, R) :- sub1(N, Nm1), fac(Nm1, S), mult(N, S, R).
rfi-p> listing
fib(0,One) :- add1(0,One).
fib(One,One) :- add1(0,One).
fib(N,R) :-
        sub1(N,Nm1),
        sub1(Nm1,Nm2),
        fib(Nm1,R1),
        fib(Nm2,R2),
        plus(R1,R2,R).
rfi-p> % Now it is unknown:
rfi-p> fac(s[s[s[0]]], X)
unknown
rfi-p> % But in facfib the clauses are still available:
rfi-p> mcd facfib
Module:  facfib
Context:
rfi-p> listing fac
fac(0,R) :- add1(0,R).
fac(N,R) :- sub1(N,Nm1), fac(Nm1,S), mult(N,S,R).
rfi-p> pause()
true
rfi-p>
rfi-p> mcd
Module:  workspace
Context: arithmetic predsucc-sym
rfi-p> mctx + facfib
Context: arithmetic predsucc-sym facfib
rfi-p> % This extension of the context gives success:
rfi-p> fac(s[s[s[0]]], X)
true
```

```
X=s[s[s[s[s[s[0]]]]]]
rfi-p> listing
fib(0,One) :- add1(0,One).
fib(One,One) :- add1(0,One).
fib(N,R) :-
        sub1(N,Nm1),
        sub1(Nm1,Nm2),
        fib(Nm1,R1),
        fib(Nm2,R2),
        plus(R1,R2,R).
rfi-p> pause()
true
rfi-p>
rfi-p> % Create a new module fib-num in memory and fill it with a copy of
rfi-p> % workspace:
rfi-p> mtell fib-num        % fib-num takes the context of workspace too
Creating new module.
rfi-p> mcd fib-num
Module:  fib-num
Context: arithmetic predsucc-sym facfib
rfi-p> mctx= predsucc-num arithmetic
Context: predsucc-num arithmetic
rfi-p> listing
fib(0,One) :- add1(0,One).
fib(One,One) :- add1(0,One).
fib(N,R) :-
        sub1(N,Nm1),
        sub1(Nm1,Nm2),
        fib(Nm1,R1),
        fib(Nm2,R2),
        plus(R1,R2,R).
rfi-p> pause()
true
rfi-p>
rfi-p>
rfi-p> %\section{4. File related commands of the module system}
rfi-p>
rfi-p>
rfi-p> % Every file related rfi-command can be used to work on the current module.
rfi-p> % Beside this there are some new commands to save and load modules and
rfi-p> % their contexts.
rfi-p>
rfi-p> % Write fib-num to a file
rfi-p> % (answer 'no', if you are asked for overwrite permission)
rfi-p> msave fib-num
Saving module fib-num in file fib-num.rfp ..
predsucc-num is unchanged, no save.
arithmetic is unchanged, no save.
rfi-p>
rfi-p> % msave writes the module and recursively all modules of the context.
```

```
rfi-p> % The context itself is written in form of 'symbol-facts',
rfi-p> % as you can see in the first lines of the generated file
rfi-p> % ('!!' is RelFun's shell escape to execute a Unix command):
rfi-p> !! cat fib-num.rfp
rfi-p>
rfi-p> % We can load fib-num and all of its submodules with a single command:
rfi-p> mdestroy --all                          % clear memory
Removing from memory: predsucc-num
Removing from memory: predsucc-sym
Removing from memory: arithmetic
Removing from memory: facfib
Removing from memory: fib-num
All user modules have been destroyed.
rfi-p>
rfi-p> % 'load' reads a module and recursively the modules written as symbol fac
rfi-p> % If some module exists already in memory they are not reloaded
rfi-p> % (principle of code sharing).
rfi-p> % It is possible to force loading with the command 'reload'.
rfi-p> load fib-num
Reading file "fib-num.rfp"
  Reading file "predsucc-num.rfp"
  Creating module predsucc-num
  Reading file "arithmetic.rfp"
  Creating module arithmetic
Creating module fib-num
rfi-p> mforest
workspace
sortbase
prelude
tracebase
predsucc-num
arithmetic
fib-num
  predsucc-num
  arithmetic
rfi-p> pause()
true
rfi-p>
rfi-p> % Evaluate a goal:
rfi-p> mctx= fib-num
Context: fib-num
rfi-p> mforest
workspace
  fib-num
    predsucc-num
    arithmetic
sortbase
prelude
tracebase
predsucc-num
```

```
arithmetic
fib-num
  predsucc-num
  arithmetic
rfi-p> fib(4,R)
true
R=5
rfi-p> pause()
true
rfi-p>
rfi-p> % It is also possible to write a flat version of fib-num to file,
rfi-p> % not containing any include files but all clauses:
rfi-p> mcreate fib-num-flat              % create an empty module
Creating new modules.
rfi-p> mcd fib-num-flat
Module:  fib-num-flat
Context:
rfi-p> % collect 'fib-num' including all of its submodules and write a flat
rfi-p> % list of clauses into the current module:
rfi-p> mflatten fib-num
Modules: fib-num predsucc-num arithmetic
rfi-p> tell fib-num-flat                 % write it to file
rfi-p> !! cat fib-num-flat.rfp
rfi-p> pause()
true
rfi-p>
rfi-p>
rfi-p> %\section{5. Short review of all module commands}
rfi-p>
rfi-p>
rfi-p> % Most of the module commands have been mentioned above.
rfi-p> % 'help' gives more details about a specific command.
rfi-p>
rfi-p> mhelp

Commands of the module system:
  mconsult     <mod1> ..     extends the current module
  mreconsult   <mod1> ..     extends the current module
  mreplace     <mod>         replaces the contents of the current module
  mtell        <mod>         copies current module to <mod>
  mcreate      <mod> ..      creates empty modules
  mdestroy     <mod1> ..     removes modules from memory
  mdestroy     --all         removes all user modules from memory
  mctx=                      clears the context
  mctx=        <mod1> ..     sets the context
  mctx         [+|-] <mod1> ..     extends/reduces the context
  mcd                        makes workspace the current module
  mcd          [<mod>]       sets the current module
  mlisting     [<pattern>]   searches along the context for <pattern>
  ml           [<pattern>]   short for mlisting
```

```
  minfo                            shows context and name of current module
  mforest      [<mod1> ..]  shows the module hierarchie
  load         <file1> ..   loads files and creates modules
  reload       <mod1> ..    loads files and creates modules
  msave        <mod1> ..    saves <mod1> .. and all of its sub-modules
  map          <rfi-cmd> <mod1> ..   executes command on the modules
  map          <rfi-cmd> --all       executes command on all modules
  mflatten     <mod1> ..    creates a flat list of clauses
  mhelp                     this overview

rfi-p> help minfo
minfo:

format: minfo

options:

effect: shows context and name of the current module.

see also: mforest, mhelp

rfi-p>
rfi-p>
rfi-p> %\section{6. Modules and the emulator}
rfi-p>
rfi-p>
rfi-p> % The module system can also be used with RelFun's compiler.
rfi-p>
rfi-p> % Remember the three areas of RelFun containing clauses and code:
rfi-p> %
rfi-p> %    -  the database of the interpreter build from the current
rfi-p> %       module and its context
rfi-p> %    -  the database of the emulator containing clauses
rfi-p> %    -  the code area of the emulator containing compiled code
rfi-p> %
rfi-p> % If the interpreter is active only its database is relevant.
rfi-p> % If the emulator is active it uses the code area to evaluate a goal.
rfi-p>
rfi-p> % It is recommended to use the compiler only in the emulator.
rfi-p> % It works in two steps:
rfi-p> %   First there are some horizontal (source-to-source) steps making
rfi-p> %   read-modify-write-cycles on the database of the emulator.
rfi-p> %   Second this database is vertically compiled to create code
rfi-p> %   to fill the code area.
rfi-p>
rfi-p> % The compiler does not know anything about contexts.
rfi-p> % Its input is a flat list of clauses. So there must be some point
rfi-p> % to take the current module and its context to create a flat list
rfi-p> % of clauses putting it into the clause-database of the emulator.
rfi-p> % This is done whenever the emulator is activated with the
```

```
rfi-p> % 'emul' command.
rfi-p>
rfi-p> % Let's demonstrate this with the facfib example:
rfi-p>
rfi-p> mdestroy --all
Removing from memory: predsucc-num
Removing from memory: arithmetic
Removing from memory: fib-num
Removing from memory: fib-num-flat
All user modules have been destroyed.
rfi-p> load facfib arithmetic predsucc-num predsucc-sym
Reading file "facfib.rfp"
Creating module facfib
Reading file "arithmetic.rfp"
Creating module arithmetic
Reading file "predsucc-num.rfp"
Creating module predsucc-num
Reading file "  7
predsucc-sym.rfp"
Creating module predsucc-sym
rfi-p> mctx= facfib arithmetic predsucc-num
Context: facfib arithmetic predsucc-num
rfi-p> fac(4, X)
true
X=24
rfi-p> pause()
true
rfi-p>
rfi-p> % Activation of the emulator.
rfi-p> emul
Collecting modules for the emulator:
sortbase workspace facfib arithmetic predsucc-num
rfe-p> compile
rfe-p> fac(4,X)
true
X=24
rfe-p>
rfe-p> % The emulator does not know pause(), so we switch back
rfe-p> % to the interpreter:
rfe-p> inter
rfi-p> pause()
true
rfi-p>
rfi-p> % The symbolic version:
rfi-p> mctx - predsucc-num + predsucc-sym
Context: facfib arithmetic predsucc-sym
rfi-p> fac(s[s[s[0]]], X)
true
X=s[s[s[s[s[s[0]]]]]]
rfi-p> emul
```

```
Collecting modules for the emulator:
sortbase workspace facfib arithmetic predsucc-sym
rfe-p> compile
rfe-p> fac(s[s[s[0]]], X)
true
X=s[s[s[s[s[s[0]]]]]]
rfe-p> inter
rfi-p> pause()
true
rfi-p>
rfi-p> % Normally every activation of the emulator destroys the clause-databa
rfi-p> % You can hold it by giving the option --nocopy:
rfi-p>
rfi-p> emul --nocopy
Database of compiler is not changed.
rfe-p> fac(s[s[s[s[0]]]], X)
true
X=s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[s[0]]]]]]]]]]]]]]]]]]]]]]]]
rfe-p> inter
rfi-p>
rfi-p>
rfi-p> %\section{Appendix}
rfi-p>
rfi-p>
rfi-p> % Now have a look how the module system works on a complicated example
rfi-p>
rfi-p>
rfi-p> %                               m1
rfi-p> %                                |
rfi-p> %                                |
rfi-p> %                          -------------
rfi-p> %                          |           |
rfi-p> %                          |           |
rfi-p> %                        m11         m12
rfi-p> %                          |           |
rfi-p> %                          |           |
rfi-p> %                          -------------
rfi-p> %                                |
rfi-p> %                                |
rfi-p> %                              m112
rfi-p>
rfi-p>
rfi-p> % First the module system breaks this DAG into a tree:
rfi-p>
rfi-p>
rfi-p> %                               m1
rfi-p> %                                |
rfi-p> %                                |
rfi-p> %                          -------------
rfi-p> %                          |           |
```

```
rfi-p> %                          |            |
rfi-p> %                         m11          m12
rfi-p> %                          |            |
rfi-p> %                          |            |
rfi-p> %                         m112         m112
rfi-p>
rfi-p>
rfi-p> % Second it traverses the tree in a pre-order way
rfi-p> % to yield a flat list of modules:
rfi-p> %
rfi-p> %   m1 m11 m112 m12 m112
rfi-p>
rfi-p>
rfi-p> % Third duplicates are removed from the right to create
rfi-p> % the final module list:
rfi-p> %
rfi-p> %   m1 m11 m112 m12
rfi-p>
rfi-p>
rfi-p> % To verify this we create the module hierarchie and use
rfi-p> % the 'emul' command to flatten the DAG:
rfi-p>
rfi-p> % Create empty modules:
rfi-p> mcreate m1 m11 m12 m112
Creating new modules.
rfi-p> mcd                                    % switch to workspace
Module:  workspace
Context: facfib arithmetic predsucc-sym
rfi-p>
rfi-p> % chain modules:
rfi-p> mctx= m1
Context: m1
rfi-p> mcd m1
Module:  m1
Context:
rfi-p> mctx= m11 m12
Context: m11 m12
rfi-p> mcd m11
Module:  m11
Context:
rfi-p> mctx= m112
Context: m112
rfi-p> mcd m12
Module:  m12
Context:
rfi-p> mctx= m112
Context: m112
rfi-p> mcd
Module:  workspace
Context: m1
```

```
rfi-p> mforest
workspace
  m1
    m11
      m112
    m12
      m112
sortbase
prelude
tracebase
facfib
arithmetic
predsucc-num
predsucc-sym
m1
  m11
    m112
  m12
    m112
m11
  m112
m12
  m112
m112
rfi-p> pause()
true
rfi-p>
rfi-p> % This shows the flat list:
rfi-p> emul
Collecting modules for the emulator:
sortbase workspace m1 m11 m112 m12
rfe-p> inter
```

G The RELFUN Prelude

```
; Active (call-by-value) tuples defined for compilability:
(ft (tup) `(tup))
(ft (tup _first | _rest) `(tup _first | _rest))

; One of the following pause clauses should always be commented out:
;(hn (pause))            ; no manual dialog steps in exec (pure batch use)
(hn (pause) (relfun)) ; manual dialog steps in exec (semi-interactive use)
```

H RELFUN file structure

H.1 Directory tree

```
.
├── RFM
│   ├── aux
│   ├── classifier
│   ├── codegenerator
│   ├── docu
│   │   ├── help
│   │   ├── papers
│   │   ├── manual
│   │   └── guide
│   ├── led
│   ├── rfi
│   ├── demo
│   │   ├── sampler
│   │   │   └── elements-mod
│   │   ├── dialog
│   │   ├── trans
│   │   ├── rtplast
│   │   │   └── orf
│   │   ├── mcad2nc
│   │   │   └── ll
│   │   ├── exa
│   │   ├── drlh
│   │   ├── findom
│   │   │   ├── domexc
│   │   │   └── fd
│   │   ├── bench
│   │   ├── extensions
│   │   ├── tcl
│   │   ├── ic
│   │   ├── modules
│   │   │   ├── recyc
│   │   │   └── facfib
│   │   ├── logvar
│   │   └── types
│   │       └── sorts
│   ├── gama
│   ├── index
│   ├── normalizer
│   ├── mode-interpreter
│   ├── tcl
│   │   └── menu
│   ├── org-patches
│   └── knownbugs
└── top
    └── demo
        ├── mcad2nc
        └── decplan
```

H.2 Detailed file listing

```
./RELFUN:
README          RFM/          init-tcl.lsp   init.lsp      top/
```

```
./RELFUN/RFM:
NEW-FEATURES              index/                   prelude.asm.PRE-GABRIEL
RFM-system.lisp           knownbugs/               prelude.asm.PRETUPOF
aux/                      led/                     prelude.asm.TUPOFERROR
classifier/               mode-interpreter/        prelude.ll
codegenerator/            normalizer/              prelude.rf
colab-interface.lisp      org-patches/             prelude.rf.old
demo/                     prelude-for-gama.rf      rfi/
docu/                     prelude-header.asm       tcl/
gama/                     prelude.asm

./RELFUN/RFM/aux:
aux.lisp      our-fs.lisp

./RELFUN/RFM/classifier:
Simplify.lisp               absyn2.lisp             classify.lisp.PRECUT
absyn-macro.lisp            clasPas1.lisp           simplify.lisp
absyn-macro.lisp.PRECUT     clasPas2.lisp
absyn1.lisp                 classify.lisp

./RELFUN/RFM/codegenerator:
absynt.lisp             cg5.lisp.PRECUT         instr.lisp.PRE-GABRIEL*
absynt.lisp.PRECUT      cgis.lisp               instr.lisp.PRECUT
assocf.lisp             cgis.lisp.put-error     misc.lisp
cg5.lisp                instr.lisp

./RELFUN/RFM/demo:
bench/                           master-lucid.18.07.96.script
descriptions.old                 master.bat
dialog/                          mcad2nc/
drlh/                            modules/
exa/                             old-master.script.commented
extensions/                      rtplast/
findom/                          sampler/
ic/                              tcl/
logvar/                          trans/
master-lucid.10.07.96.script     types/

./RELFUN/RFM/demo/bench:
bench.bat    fac.rf     hgh.rf     nrev.rf
fac.bat      hgh.bat    nrev.bat

./RELFUN/RFM/demo/dialog:
brief-intro.rf@   cmlist.rf      fibcut.rf      lazydiff.rf
callex.rf         dialog.bat     instex.rf      mapper.rf

./RELFUN/RFM/demo/drlh:
drlh.bat          emul.rf        path.rf        unpack.rf
drlh.script       load-all.bat   predrlh.rf
e-load-all.bat    norm.rf        proc.rf
```

```
e-tst.script    partslist.rf    samples.rf

./RELFUN/RFM/demo/exa:
PALIN                          palin.asm
PALIN.ps                       palin.bat
akk.bat                        palin.rf
akk.rf                         palin.rfp
analogy-compilable.rf          palin.script
bench.rf                       palin.script2
bench1.rf                      partrf.bat
bench2.rf                      partrf.rfp
conventional.bat               qsort.rf
conventional.script            quick.rf
conventional.script.html       syntax-test.bat
counttree.bat                  test.extend.l
counttree.rfp                  test.l
demo.rf                        test.rf
demo1.rf                       test2.rf
demo2.rf                       testinst.rf
demo3.rf                       wang.rf
demo4.rf                       wangaux.rf
demo5.rf                       wangtree.rf
demo6.rf                       workpiece
exa.bat                        workpiece-tup-lhs-flat.rf
example2.rf                    workpiece.bat
fehler.rf                      workpiece.rf
fun6.rf                        workpiece.script
fuzzy.rfp                      wp-demo-bin.rf
gcd.rf                         wp-demo-bin2.rf
occur-check.rf                 wp-demo.rf
palin-1storder.rfp             wp-demo2.rf
palin-multi-order.bat          wpnorm.rf
palin-multi-order.script

./RELFUN/RFM/demo/extensions:
MegaTest.Scr             gser.rfp                  serialisef.rfp
MegaTest.Script          inv.rf                    serialiseg.rfp
attval.rf                inv.rfp                   serialiseg.rfp.pair
attval.rfp               lisp.rf                   serialiseg.script
extensions.bat           lisp.rfp                  serialiser.rf
extensions.bat.old       lispeval.rfp              serialiser.rfp
extensions.dialog        mapcar.class              sertime.bat
extensions.dialog.old    mu-operator.rf            sertime.gser
extensions.script.old    mu-operator.rfp           sertime.gserll
extensions1.bat          quicksort.rfp             sertime.gserqsort
extensions2.bat          revise.rf                 sertime.serialgf
extensions3.bat          revise.rfp                sertime.serialiseg
extensions4.bat          serial.rfp                sertime.txt
extensions5.bat          serialgf.rfp              signum.rf
extensionsx.bat          serialise-complexity      signum.rfp
```

```
genints.rf             serialise-demo.bat       wang.rf
genints.rfp            serialise-demo.script    wang.rfp
gser-qsort.rfp         serialise.handout        wangaux.lisp
gser.bat               serialise.handout13.ps   wangtree.rf
gser.bat.old           serialise.rfp            wangtree.rfp
gser.lisp              serialisef.rf            wangwork.rfp

./RELFUN/RFM/demo/findom:
domexc/  fd/

./RELFUN/RFM/demo/findom/domexc:
bndtest.bat           domexc.bat.old          domexc.script.uklirb
domexc.bat            domexc.script           domexc1.script

./RELFUN/RFM/demo/findom/fd:
fd-exa.rf        fd.old-once.rf    findom.bib
fd.bat           fd.rf

./RELFUN/RFM/demo/ic:
ic-fun.bat        ic-rel-fun.fol1   ms-time.rfp
ic-fun.rfp        ic-rel-fun.fol2   time.rfp
ic-fun.script     ic-rel.rfp        time.script

./RELFUN/RFM/demo/logvar:
avereplace.rfp   logvar.script    optcast.rfp
logvar.bat       maxtree.rfp      prodcast.rfp

./RELFUN/RFM/demo/mcad2nc:
anc-program.rf   examples.rf      mcad2nc.bat      skeletal.rf
class-feat.rf    library.rf       mcad2nc.script
demo.rf          ll/              rng2p.rf

./RELFUN/RFM/demo/mcad2nc/ll:
anc-program.rf   examples.rf      mcad2nc.bat      skeletal.rf
class-feat.rf    library.rf       mcad2nc.script
demo.rf          ll.script        rng2p.rf

./RELFUN/RFM/demo/modules:
facfib/  recyc/

./RELFUN/RFM/demo/modules/facfib:
arithmetic.rfp       module-demo.bat       predsucc-num.rfp
facfib.rfp           module-demo.script    predsucc-sym.rfp

./RELFUN/RFM/demo/modules/recyc:
a4-folien.sty                 konzepte.tex
dateien-im-filesystem.tex     matmod.script
hochformat.aux                mit-modul-system.tex
hochformat.dvi                precious.rfp
hochformat.log                precious.tex
```

```
hochformat.tex               querformat.aux
jewel.rfp                    querformat.dvi
jewel.tex                    querformat.log
konzepte-an-hb.aux           querformat.tex
konzepte-an-hb.dvi           recyc-auto.rfp
konzepte-an-hb.log           recyc-auto.tex
konzepte-an-hb.tex           recyc-car.rfp
konzepte-sun.aux             recyc-electrics.rfp
konzepte-sun.dvi             recyc-electrics.tex
konzepte-sun.log             tmp.dvi
konzepte-sun.ps              tmp.tex
konzepte-sun.tex

./RELFUN/RFM/demo/rtplast:
aux.rfp                rtp-sort.bat          rtplast-taxo-sub.rf
orf/                   rtplast-inf.rf

./RELFUN/RFM/demo/rtplast/orf:
rtplast.bat   rtplast.rfp

./RELFUN/RFM/demo/sampler:
analogy.bat                  facfix.bat
analogy.rf                   facfix.rf
analogy.rf.mysterioes        geoparse.bat
brief-intro.bat              geoparse.rf
brief-intro.rf               geoparsehn.rf
brief-intro.rfp              geoparsel.rf
brief-intro.script           geoparser.rf
elements-mod/                prime.bat
elements-new.bat             prime.rfp
elements.bat                 sampler.bat
elements.rfp                 sampler.script
elements.rfp.html            sampler.script.alt
elements.script              solid.bat
elements.script.PRE-UMBAU    solid.rf
elements.script.html         sort.bat
engin-know.bat               sort.rf
engin-know.rf                sort.rfp
eppler.rf                    wpnorm-non-ground.rf
erathostenes.bat             wpnorm.bat
erathostenes.bat.mc          wpnorm.rf
erathostenes.rfp

./RELFUN/RFM/demo/sampler/elements-mod:
README                     elements-groups.rfp       elements-utility.rfp
elements-access.rfp        elements-loader.rfp       elements-validation.rfp
elements-atm.rfp           elements-mod.bat

./RELFUN/RFM/demo/tcl:
castles4.rfp      queec4.bat        queec4.rfp        tcl-prelude.rfp
```

```
./RELFUN/RFM/demo/trans:
cext.bat                lisp-evaluate.bat      prop-log.script
cext.script             lisp-evaluate.rf       share.bat
cext.script.alt         passtuptest.bat        share.rf
demostruc.bat           passtuptest.rf         trans.bat
demostruc.rf            passtuptest.script     trans.script
demostruc.script        prop-log.bat           tupcns.bat
demostruc.script.alt    prop-log.rf            tupcns.rf

./RELFUN/RFM/demo/types:
bndtst.bat                    sorts/
buisob.bat                    types.bat
deanon.bat                    types.diff
dombnd.bat                    types.script.whith-new-rfi
dombnd.rf                     types.script.whith-old-rfi
domexc.bat                    typin.bat
dyn-sg.bat                    typin.script
exc.bat                       typin.script.with-new-rfi
instant.bat                   typin.script.with-old-rfi
instant.script

./RELFUN/RFM/demo/types/sorts:
exa1.rfp          glb.rfp             pet-deutsch.bat     pet.script
exa4.rfp          pet-base-sub.rfp    pet.bat

./RELFUN/RFM/docu:
README   guide/   help/    manual/  papers/

./RELFUN/RFM/docu/guide:
93.2.bib@                   conventional.dialog
95.1.bib@                   dfkititle.tex
96.1.bib@                   dfkititlepage.sty
RFM-Guide.aux               dir-tree.tex
RFM-Guide.bbl               dir-tree1.tex
RFM-Guide.blg               drl+l.ps
RFM-Guide.dvi               dyn-sg.dialog
RFM-Guide.log               epsfigure.tex
RFM-Guide.ps                ls-RC.tex
RFM-Guide.tex               module-demo.dialog
RFM-Guide.toc               own.bib@
RFM-cmds.tex                pet.dialog
brief-intro.tex             prelude.tex
buisob.dialog               rf-ascii.tex
comdefs.tex                 rf.short
commands.tex                startup.tex
commands.tex.old-version    typin.dialog

./RELFUN/RFM/docu/help:
a0.tex                  indexing.tex            reconsult.tex
```

```
a0ft.tex                init.lisp               relfun.tex
a0hn.tex                inter.tex               reload.tex
asm.tex                 l.tex                   replace.tex
assem.tex               lconsult.tex            rf2rf.tex
az.tex                  lisp.tex                rf2rfp.tex
azft.tex                listclass.tex           rfp2rf.tex
azhn.tex                listcode.tex            rfp2rfp.tex
bal2bap.tex             listing.tex             rx.tex
bap2bal.tex             load.tex                rxft.tex
break.tex               lreplace.tex            rxhn.tex
browse-sortbase.tex     m.tex                   script.tex
builtins.tex            map.tex                 showdepth.tex
bye.tex                 mcd.tex                 sl.tex
classify.tex            mconsult.tex            sortbase.tex
codegen.tex             mcreate.tex             sp.tex
compile-sortbase.tex    mctx.tex                spy.tex
compile.tex             mctx=.tex               strict-sortbase.tex
complete-sortbase.tex   mdestroy.tex            style.tex
consult-sortbase.tex    mflatten.tex            tell.tex
consult.tex             mforest.tex             texhelpfiles
deanon.tex              mhelp.tex               texput.log
destroy-sortbase.tex    minfo.tex               timermode.tex
destroy.tex             miser-level.tex         trace.tex
emul.tex                mlist.tex               uncomma.tex
endscript.tex           more.tex                unique-sortbase.tex
exec.tex                mreconsult.tex          unlambda.tex
hash.tex                mreplace.tex            unmacro.tex
help.tex                msave.tex               unor.tex
help.tex.bak            mtell.tex               untrace.tex
help2dvi                nospy.tex               untype.tex
help2dvi.tex            ori.tex                 verti.tex
hitrans.tex             prelude.tex
horizon.tex             print-width.tex

./RELFUN/RFM/docu/manual:
93.2.bib@          RFM-Manual.blg   RFM-Manual.ps2   comdefs.tex
95.1.bib@          RFM-Manual.dvi   RFM-Manual.tex   lp.bib@
RFM-Manual.aux     RFM-Manual.log   RFM-Manual.toc   own.bib@
RFM-Manual.bbl     RFM-Manual.ps    RFM-cmds.tex     wam.bib

./RELFUN/RFM/docu/papers:
bad-honnef.ps.gz   index.ps.gz

./RELFUN/RFM/gama:
README                           gwam.before-feat.lisp
buisob.lisp.NOW-IN-RFI.LISP      gwam.lisp
deta.bat                         gwam.lisp.PRE-GABRIEL
deta.bat.old                     gwam.lisp.PRESUBSUMES
deta.obj                         gwaminit.asm
deta.ps                          ll.lisp
```

```
deta.script                 llama.lisp
emul-types.txt              new-once.rf
fw-asm.lisp                 new-once.txt
gasm-eval.lisp              old-gwam.lisp
gasm.lisp                   orf2.lisp
gaux.lisp                   rf2ll+.lisp
gcla.lisp                   rf2ll.lisp
gcompile.lisp               subsume-patch.lisp
gdestroy.lisp               tbox.lisp
ginit.lisp                  texa.bat
gmem.lisp                   texa.script
gmht.lisp                   typserv.lisp
gwam-with-cut.lisp          typserv.lisp.PRE-BUISOB
gwam-with-cut.readme        typserv.lisp.old

./RELFUN/RFM/index:
README            icg.lisp          idx.lisp          linear.lisp
cg5-patch.lisp    icl.lisp          iif.lisp          misc-patch.lisp

./RELFUN/RFM/knownbugs:
README           rfm-agenda      tecvoc.readme
knownbugslist    tecvoc.lisp     tecvoc.rfp

./RELFUN/RFM/led:
ansi.lisp   led.lisp    lex.lisp

./RELFUN/RFM/mode-interpreter:
mode-interpreter.lisp     mode-rfi-interface.lisp

./RELFUN/RFM/normalizer:
debug.lisp        normalizer.lisp

./RELFUN/RFM/org-patches:
CUT-patches.lisp     cd.patch              more-patches.lisp
cd.diff              cd.readme             patches.12.94.lisp
cd.new               ll-patches.lisp

./RELFUN/RFM/rfi:
patches.lisp         rfi.lisp              sortbrowser.lisp    tracer.rfp
prelude.rf@          rfi.sbin.previous     start.lisp          tracer.script
relfun               rfmrc.lisp            syntra.lisp
relfun.h             solvao.rf             tracer-tst.rfp
relfun.lisp          solverf.rf            tracer.bat

./RELFUN/RFM/tcl:
box_empty.xpm          filebrowser.tcl          sortbrowser.tcl
box_full.xpm           gen2.tcl                 turtle.bit
cliccdrl*              menu/                    turtle2.bit
construction.xpm       relfun.tcl               turtle3.bit
drl*                   relfunProcedures.tcl
```

```
./RELFUN/RFM/tcl/menu:
moduleEdgeMenu.emn   moduleIconMenu.imn   moduleViewMenu.vmn

./RELFUN/top:
defsystem.lisp            our-fs.bsp2
defsystem.lisp.PRE-CLISP  our-fs.lisp
demo/                     start-RFM.lisp
our-fs.bsp

./RELFUN/top/demo:
decplan/  mcad2nc/

./RELFUN/top/demo/decplan:
abi-decplan.bat       ext-decplan.bat
decplan2.bat          feature-concepts.tx

./RELFUN/top/demo/mcad2nc:
MICRO.HI              feat2p.rf                  skeletal.rf
aa-mcad.bat           hybrid.rf                  start.bat
anc-program.rf        init.lisp                  starttxfw.bat
berlin.bat            inst.ctx                   tools.ctx
brief-intro.rf        library.rf                 tx-access.rf
class-feat.rf         micro.tx                   tx-additions.rf
constdef.lisp         micro2.tx                  tx-parts-emul.rf
demo.rf               rfm-only-mcad2nc.script    wp-mcad2nc.rf
demotxfw.rf           rfm-only.bat               wp3.rf
examples.rf           rng2p.rf
```

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

-Bibliothek, Information
und Dokumentation (BID)-
PF 2080
67608 Kaiserslautern
FRG

Telefon (0631) 205-3506
Telefax (0631) 205-3210
e-mail
dfkibib@dfki.uni-kl.de
WWW
http://www.dfki.uni-sb.de/dfkibib

Veröffentlichungen des DFKI

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder (so sie als per ftp erhaeltlich angemerkt sind) per anonymous ftp von ftp.dfki.uni-kl.de (131.246.241.100) im Verzeichnis pub/Publications bezogen werden. Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or (if they are marked as obtainable by ftp) by anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) in the directory pub/Publications.
The reports are distributed free of charge except where otherwise noted.

DFKI Research Reports

1996

RR-96-05
Stephan Busemann
Best-First Surface Realization
11 pages

RR-96-03
Günter Neumann
Interleaving
Natural Language Parsing and Generation
Through Uniform Processing
51 pages

RR-96-02
E.André, J. Müller , T.Rist:
PPP-Persona: Ein objektorientierter Multimedia-Präsentationsagent
14 Seiten

1995

RR-95-20
Hans-Ulrich Krieger
Typed Feature Structures, Definite Equivalences, Greatest Model Semantics, and Nonmonotonicity
27 pages

RR-95-19
Abdel Kader Diagne, Walter Kasper, Hans-Ulrich Krieger
Distributed Parsing With HPSG Grammar
20 pages

RR-95-18
Hans-Ulrich Krieger, Ulrich Schäfer
Efficient Parameterizable Type Expansion for Typed Feature Formalisms
19 pages

RR-95-17
Hans-Ulrich Krieger
Classification and Representation of Types in TDL
17 pages

RR-95-16
Martin Müller, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-15
Joachim Niehren, Tobias Van Roy
Title not set
0 pages

Note: The author(s) were unable to deliver this document for printing before the end of the year. It will be printed next year.

RR-95-14
Joachim Niehren
Functional Computation as Concurrent Computation
50 pages

RR-94-34
Stephan Busemann, Stephan Oepen, Elizabeth A. Hinkelman,
Günter Neumann, Hans Uszkoreit
COSMA - Multi-Participant NL Interaction for Appointment Scheduling
80 pages

RR-94-33
Franz Baader, Armin Laux
Terminological Logics with Modal Operators
29 pages

RR-94-31
Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

RR-94-23
Gert Smolka
The Definition of Kernel Oz
53 pages

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz
Encapsulated Search and Constraint Programming in Oz
21 pages

RR-94-19
Rainer Hoch
Using IR Techniques for Text Classification in Document Analysis
16 pages

RR-94-18
Rolf Backofen, Ralf Treinen
How to Win a Game with Features
18 pages

RR-94-17
Georg Struth
Philosophical Logics—A Survey and a Bibliography
58 pages

RR-94-16
Gert Smolka
A Foundation for Higher-order Concurrent Constraint Programming
26 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr
Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-13
Jana Koehler
Planning from Second Principles—A Logic-based Approach
49 pages

RR-94-12
Hubert Comon, Ralf Treinen
Ordering Constraints on Trees
34 pages

RR-94-11
Knut Hinkelmann
A Consequence Finding Approach for Feature Recognition in CAPP
18 pages

RR-94-10
Knut Hinkelmann, Helge Hintze
Computing Cost Estimates for Proof Strategies
22 pages

RR-94-08
Otto Kühn, Björn Höfling
Conserving Corporate Knowledge for Crankshaft Design
17 pages

RR-94-07
Harold Boley
Finite Domains and Exclusions as First-Class Citizens
25 pages

RR-94-06
Dietmar Dengler
An Adaptive Deductive Planning System
17 pages

RR-94-05
Franz Schmalhofer, J. Stuart Aitken, Lyle E. Bourne jr.
Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-03
Gert Smolka
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-02
Elisabeth André, Thomas Rist
Von Textgeneratoren zu Intellimedia-Präsentationssystemen
22 Seiten

RR-94-01
Elisabeth André, Thomas Rist
Multimedia Presentations: The Support of Passive and Active Viewing
15 pages

D-95-07
Ottmar Lutzy
Morphic - Plus
Ein morphologisches Analyseprogramm für die deutsche Flexionsmorphologie und Komposita-Analyse
74 pages

D-95-06
Markus Steffens, Ansgar Bernardi
Integriertes Produktmodell für Behälter aus Faserverbundwerkstoffen
48 Seiten

D-95-05
Georg Schneider
Eine Werkbank zur Erzeugung von 3D-Illustrationen
157 Seiten

D-95-04
Victoria Hall
Integration von Sorten als ausgezeichnete taxonomische Prädikate in eine relational-funktionale Sprache
56 Seiten

D-95-03
Christoph Endres, Lars Klein, Markus Meyer
Implementierung und Erweiterung der Sprache $\mathcal{ALCP}$
110 Seiten

D-95-02
Andreas Butz
BETTY
Ein System zur Planung und Generierung informativer Animationssequenzen
95 Seiten

D-95-01
Susanne Biundo, Wolfgang Tank (Hrsg.)
PuK-95, Beiträge zum 9. Workshop „Planen und Konfigurieren", Februar 1995
169 Seiten

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

1994

D-94-15
Stephan Oepen
German Nominal Syntax in HPSG
— On Syntactic Categories and Syntagmatic Relations —
80 pages

D-94-14
Hans-Ulrich Krieger, Ulrich Schäfer
TDL - A Type Description Language for HPSG, Part 2: User Guide.
72 pages

D-94-12
Arthur Sehn, Serge Autexier (Hrsg.)
Proceedings des Studentenprogramms der 18. Deutschen Jahrestagung für Künstliche Intelligenz KI-94
69 Seiten

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.)
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 pages

Note: This document is no longer available in printed form.

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.)
Working Notes of the 1994 International Workshop on Description Logics
118 pages

Note: This document is available for a nominal charge of 25 DM (or 15 US-$).

D-94-09
Technical Staff
DFKI Wissenschaftlich-Technischer Jahresbericht 1993
145 Seiten

D-94-08
Harald Feibel
IGLOO 1.0 - Eine grafikunterstützte Beweisentwicklungsumgebung
58 Seiten

D-94-07
Claudia Wenzel, Rainer Hoch
Eine Übersicht über Information Retrieval (IR) und NLP-Verfahren zur Klassifikation von Texten
25 Seiten

D-94-06
Ulrich Buhrmann
Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien
117 Seiten

D-94-04
Franz Schmalhofer, Ludger van Elst
Entwicklung von Expertensystemen: Prototypen, Tiefenmodellierung und kooperative Wissensevolution
22 Seiten

D-94-03
Franz Schmalhofer
Maschinelles Lernen: Eine kognitionswissenschaftliche Betrachtung
54 Seiten

Note: This document is no longer available in printed form.

D-93-12
Document

RELFUN Guide: Programming with Relations and Functions Made Easy

(Second, Revised Edition)

Harold Boley,
Simone Andel, Klaus Elsbernd, Michael Herfert,
Michael Sintek, Werner Stein