Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Technical
Memo
TM-94-03

Uncertainty-Valued Horn Clauses

Victoria Hall

September 1994

Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
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, and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for 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

Uncertainty-Valued Horn Clauses

Victoria Hall

DFKI-TM-94-03

ISSN 0946-0071

Uncertainty-Valued Horn Clauses

Victoria Hall

DFKI
Postfach 2080
67608 Kaiserslautern
Germany

September 6, 1994

Abstract

There are many forms of uncertainty, each usually again having more than one theoretical model.

Therefore, a very flexible kind of uncertainty-valued Horn clauses is introduced in RELFUN in section 1. They have a head, several premises and an uncertainty factor, which represents the uncertainty of the clause. The premises are all 'functional' in the sense that their returned value is again an uncertainty value. These premises and the uncertainty factor of an uncertainty rule become embedded into the arguments of a combination function when translating uncertainty clauses into footed clauses (non-ground, non-deterministic functions in RELFUN, which can then be compiled as usual). The combination function can be modified by the user. It may be a built-in or a user-defined function, either of which may be computed as the value of a higher-order function.

In section 2, an application of uncertainty clauses to the uncertain concept of a 'pet holder', according to German law, is described. This and another example are then fully demonstrated in appendix A. Finally, appendix B gives a listing of the complete uncertainty translator in LISP.

Contents

1 The Uncertainty Translator **2**
- 1.1 The Uncertainty Structure 2
- 1.2 The Combination Function 3

2 Example 'Pet Holder' **5**
- 2.1 The Features 5
- 2.2 The Combination Functions 6
- 2.3 Finding a Pet Holder 7

A An Application of Uncertainty in RELFUN **9**

B The Uncertainty Translator **30**

1 The Uncertainty Translator

This section shows a simple transformational extension of RELFUN for handling uncertainty. It is based on a proposal by Harold Boley and a lot of implementational help by Michael Sintek.

1.1 The Uncertainty Structure

There are many forms of uncertainty; for example there is vagueness (fuzziness in particular), probability, and so on [4]. Furthermore there are a lot of theoretical models to simulate the various kinds of uncertainty. To stay flexible in RELFUN [1], no special model is chosen here. It is left to the user to decide in which way he wants to describe the uncertainty of his particular problem. For this, uncertainty clauses or 'uc clauses' are introduced (we show RELFUN's two syntax styles, with the focus on the former) :

Lisp style:

```
(uc (c ..)  (ucfb1 ..)  ...  (ucfbM ..)  UC_FACTOR)
```

Prolog style:

```
c(..) :-# ucfb1(..), ... ,ucfbM(..), UC_FACTOR.
```

The premises `(ucfb1 ..), ... ,(ucfbM ..)` are functional and their returned values represent uncertainty factors. The explicit uncertainty factor, UC_FACTOR, stands for the uncertainty of the rule itself. Normally, and later on in this paper, uncertainty factors will be numerical values in [0..1]. But it is also possible to think of qualitative or quantitative values in a symbolic way, for which there are two possible treatments:

- Translate symbolic values into numerical values in [0..1].
  Calculate with these numerical values in the 'normal' way.
  Retranslate numerical values into symbolic values.

- Use a special combination table for the symbolic values.

Neither the compiler nor the interpreter of normal RELFUN can directly handle uncertainty clauses. So they have to be translated into (non-ground, non-deterministic) footed clauses by embedding the premises and the uncertainty factor into a combination function, `combrule`:

Lisp style:

```
(ft (c ..)  (combrule (ucfb1 ..)  ...  (ucfbM ..)  UC_FACTOR))
```

Prolog style:

```
c(..)  :- combrule(ucfb1(..), ...  ,ucfbM(..), UC_FACTOR).
```

1.2 The Combination Function

As said before, the premises have to be functional; otherwise the combination function will produce an error. Premises defined by hornish clauses have to be changed into premises defined by footed clauses with a returned value in [0..1]. This is performed by the RELFUN command `footer`, translating all hornish definitions into footed ones. `footer` expects an argument which represents the returned value of the constructed footed clauses. By default hornish clauses are understood as certain, thus their uncertainty factor is set to 1: `footer` acts like `footer 1`

The translation of uncertainty clauses into footed clauses (as described in section 1.1) is done with the function `uncertain`. Like `footer` it translates a whole database. `uncertain` expects the combination function `combrule` as argument, which may be a built-in or a user-defined function, either of which may be computed as the value of a higher-order function.

- `uncertain min` where `min` is a built-in function,

- `uncertain myrule` where `myrule` is a user-defined function,

- `uncertain (cr)` where `cr` is a (parameterless) higher-order function which returns a combination function (like a globally declared constant, this returned value can be changed easily). This can e.g. be useful in the 'test phase' of a system, when combination functions are changed several times.

Altogether, the translation yields `ft clauses`, which can then be compiled as usual:

$$\left.\begin{array}{l} \mathsf{hn} \xrightarrow{\mathsf{footer}} \mathsf{ft} \\ \\ \mathsf{uc} \xrightarrow{\mathsf{uncertain}} \mathsf{ft} \end{array}\right\} \begin{array}{c}\textsf{compiled}\\ \textsf{into}\\ \textsf{WAM}\\ \textsf{code}\end{array}$$

Even more than one combination function can be used in a single database. For that, all uncertainty clauses whose returned value is calculated the same way are added to the database. This is then translated with the chosen combination function. After that the next uncertainty clauses are asserted or consulted, and a second translation with another combination function follows, etc. This is no problem because the `uncertain` command only changes the uncertainty clauses, while footed clauses stay unmodified.

Note that uncertainty facts have no premises: `(uc (c ..) UC_FACTOR)` and no combination function is called to calculate uncertainty values: `(ft (c ..) UC_FACTOR)`.
So, if there are only uncertainty facts, it is insignificant which combination function is given to the `uncertain` command. Actually, it is possible to write uncertainty facts directly as footed facts if the returned values are understood as an uncertainty values.

We should note that our combination function combines values of the premises of a conjunction, hence could be called AND-combination function. Uncertainty systems often also permit what we may call an OR-combination function. However there is a restriction in handling uncertainty in backtracking languages like PROLOG or RELFUN: if there exists more than one solution for a query, then the individual solutions cannot easily be (OR-) combined into a final result (see the 'certainty factors' used in MYCIN). But the observation that most of these OR-combined models are mathematically inconsistent anyway [3] makes this restriction appear an advantage. Only by using `bagof` or `tupof` , is it possible to deduce all solutions in our implementation, and then compute the solution with the greatest uncertainty factor of this collection of solutions (see function `fetch` [5] or `reduce` applied to `maxp`).

2 Example 'Pet Holder'

This section formalizes the realistic example of [2] with RELFUN's uncertainty-valued Horn clauses.

In German law, the *owner* of a pet is not always liable for the damage caused by his pet. The *pet holder*, who can be the owner, the horseman, the buyer, the finder of a deserted pet, and so on, is responsible. The problem is, that there is no *sharp* definition of 'pet holder', so a judge has to decide with common sense who is the 'pet holder'.

2.1 The Features

The term 'pet holder' is described by a set of attributes. These can be brought together in a 'deduction tree'.

To begin with, there are five natural language rules:

1. To the same extent as a person meets the costs of maintenance , OR has the advantage of use, he has an immediate interest in the pet.

2. To the same extent as a person gains profit, AND/OR loss, the person has a long term interest in the pet.

3. To the same extent as a person has an immediate, AND long term interest in the pet, he has an interest in the pet.

4. To the same extent as a person has possession of, OR is the master of the pet, he takes responsibility for the pet.

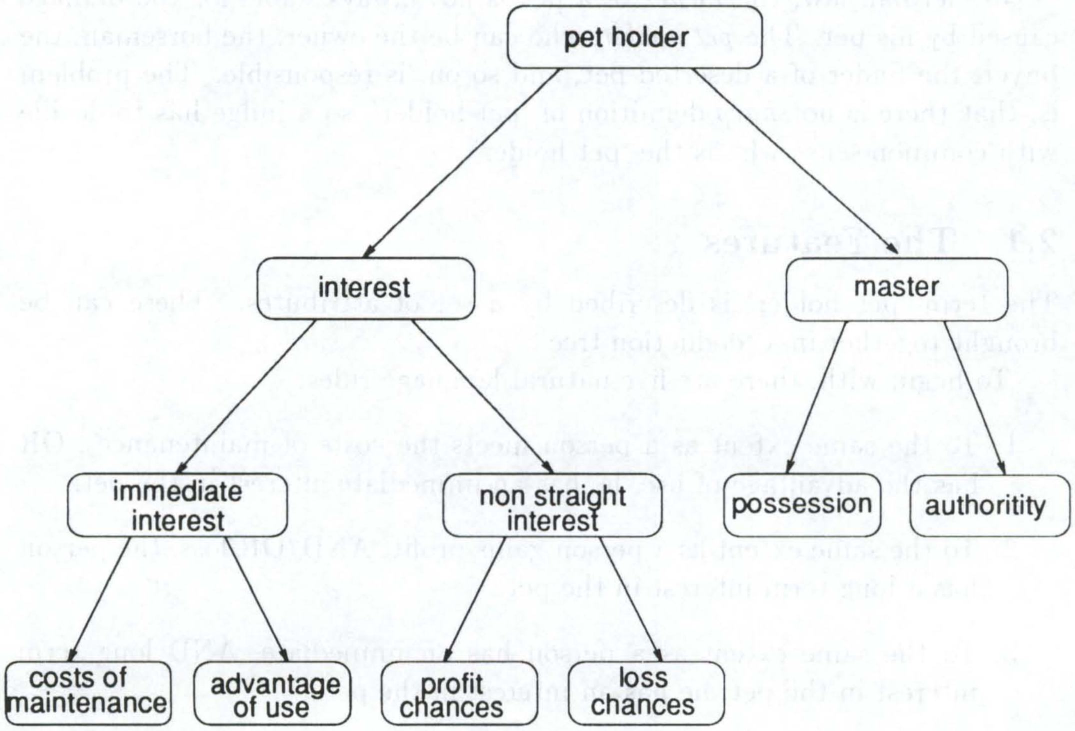
5. To the same extent as a person has interest in, AND carries responsibility for the pet, he is a pet holder.

These features are measured in terms like:

- Costs of maintenance: shelter, food, nurture, doctor etc.

- Advantage of use: receipts of work, hiring etc.

* Profit chances: costs of procuring.
* Loss chances: initial costs, costs of education etc.
* Possession: actual power, use, shelter, nurture in the own household or farm
* Master: decision about the existence of the pet (life and use)



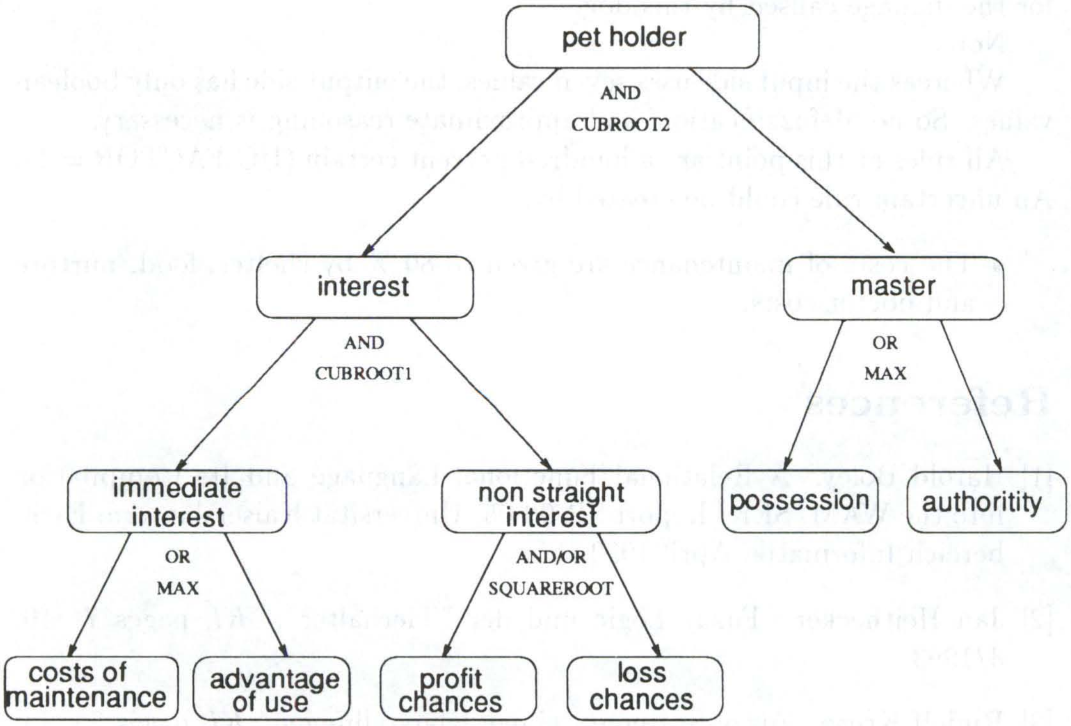
2.2 The Combination Functions

In this system more than one combination function is used. They were found by experience. It showed up, that there are four cases:

* normal AND
* normal OR
* AND with compensation

- OR with compensation

The normal AND/OR is realized with **min/max**; the AND with compensation by geometrical mean ($\sqrt[2]{x_1 \cdot \ldots \cdot x_n}$). In the case of OR with compensation it is more difficult: the attributes are in a special relation. The immediate to the long term interest are in the ratio 1 to 2 (`cubroot1` $= \sqrt[3]{x_1 \cdot x_1 \cdot x_2}$); interest to responsibility is in the inverse ratio 2 to 1 (`cubroot2` $= \sqrt[3]{x_1 \cdot x_2 \cdot x_2}$).



2.3 Finding a Pet Holder

Because it is more difficult for a human being to express something in numbers, the values for the attributes are given in natural language; they have to be translated: null = 0.01; very few = 0.1; few = 0.25; medium = 0.5; high = 0.75; very high = 0.9; total = 1. The output has to be interpreted: values over or equal to 0.5 mean 'the person is a pet holder' and values less than 0.5 mean 'the person is not a pet holder'.

E.g., it is now possible to describe the finder of a dog:

costs of maintenance	few
advantage of use	medium
profit chances	medium
loss chances	medium
possession	high
master	medium

For these values the finder of a dog is a pet holder and therefore liable for the damage caused by this dog.

Note:

Whereas the input side uses seven values, the output side has only boolean values. So no 'defuzzification' and approximate reasoning is necessary.

All rules at this point are a hundred percent certain (UC_FACTOR = 1). An uncertain rule could be created by:

- The costs of maintenance are given to 80 % by shelter, food, nurture and doctor costs.

References

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

[2] Jan Heithecker. Fuzzy Logic und der "Tierhalter". *KI*, pages 7–10, 3/1993.

[3] Rudolf Kruse. Fuzzy-Systeme - einige Klarstellungen. *KI*, pages 73, 74, 4/1993.

[4] Rudolf Kruse, Erhard Schwenk, and Jochen Heinsohn. *Uncertainty and Vagueness in Knowledge Based Systems, Numerical Methods.* Springer - Verlag, 1991.

[5] Rick LeFaivre. *Fuzzy Reference Manual.* Computer Science Department, Rutgers University, March 1977.

A An Application of Uncertainty in RELFUN

```
rfi-l> style lisp
rfi-l> inter
rfi-l> destroy
rfi-l>
rfi-l> ;****************************************************************
rfi-l> ;With this translator a new structure is defined. It is called
rfi-l> ;uncertainty structure: (uc (c...) (ucfb1...) ... (ucfbM ...)
rfi-l> ;                                                    UC-FACTOR)
rfi-l> ;The premises are functional and their return value represents
rfi-l> ;an uncertainty factor in [0 .. 1] ((ucfb1) .. (ucfbM)).
rfi-l> ;The function uncertain translates all uncertainty clauses of
rfi-l> ;the relfun database *rfi-database* into footed clauses whose
rfi-l> ;return values are the uc factors calculated with a combination
rfi-l> ;rule applied to  the uc factors of the rules and of the premises.
rfi-l> ;All other structures (hornish  and footed clauses) remain
rfi-l> ;unmodified.
rfi-l> ;
rfi-l> ;***************FACTS******************************************
rfi-l> ;
rfi-l> ;****************************************************************
rfi-l> ;                    bad_weather
rfi-l> ;                     /     \
rfi-l> ;                    /       \
rfi-l> ;                 storm      cold
rfi-l> ;                 / \        / \
rfi-l> ;                /   \      /   \
rfi-l> ;             rain   windy snow  no_sun
rfi-l> ;The relations 'snow', 'rain', etc. refer to one day (24 hours)
rfi-l> ;snowing, raining, etc.
rfi-l> ;****************************************************************
rfi-l>
rfi-l> ;Nothing happens with hornish and footed clauses!
rfi-l>
rfi-l> ;hn facts
```

```
rfi-l> az (hn (snow 12.10.92))
rfi-l> az (hn (snow 12.24.92))
rfi-l> az (hn (no_sun 10.10.92))
rfi-l>
rfi-l> ;ft facts
rfi-l> az (ft (no_sun 6.10.92) 0.4)
rfi-l> az (ft (rain 6.10.92) 0)
rfi-l> az (ft (rain 10.10.92) 0.5) ;could mean half day rain
rfi-l>
rfi-l> uncertain min
rfi-l> l
(hn (snow 12.10.92))
(hn (snow 12.24.92))
(hn (no_sun 10.10.92))
(ft (no_sun 6.10.92)
    0.4 )
(ft (rain 6.10.92)
    0 )
(ft (rain 10.10.92)
    0.5 )
rfi-l> ;PROLOG Style analogously
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;uc facts
rfi-l>
rfi-l> az (uc (rain 11.11.92) 0.13)  ; 3 hours rain on 11. nov 1992
rfi-l>
rfi-l>
rfi-l> ;Translate uncertain clauses into footed clauses with the given
rfi-l> ;uc factor as return value.
rfi-l> ;Since this is a fact and no premises are given, the uc factor
rfi-l> ;needs no calculation and stays unchanged.
rfi-l>
rfi-l> uncertain min
rfi-l> l rain
(ft (rain 6.10.92)
    0 )
```

```
(ft (rain 10.10.92)
    0.5 )
(ft (rain 11.11.92)
    0.13 )
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;PROLOG Style
rfi-l> sp
rfi-p> l rain
rain(6.10.92) :-& 0.
rain(10.10.92) :-& 0.5.
rain(11.11.92) :-& 0.13.
rfi-p> pause()
rfi-p>  bye
true
rfi-p> sl
rfi-l> ;*************RULES******************************************
rfi-l> ;hornish and footed rules stay unchanged
rfi-l>
rfi-l> ;hn rules
rfi-l> az (hn (cold _x) (snow _x))
rfi-l> az (hn (storm _x) (rain _x) (< (windforce _x) 6))
rfi-l>
rfi-l> ;ft rules
rfi-l> az (ft (storm _x) (windy _x) (/ (windforce _x) 12))
rfi-l> az (ft (storm _x) (rain _x) (< (windforce _x) 6))
rfi-l>
rfi-l> uncertain *
rfi-l>
rfi-l>
rfi-l> l cold
(hn (cold _x)
    (snow _x) )
rfi-l>
rfi-l>
rfi-l> l storm
(hn (storm _x)
```

```
    (rain _x)
    (< (windforce _x) 6) )
(ft (storm _x)
    (windy _x)
    (/ (windforce _x) 12) )
(ft (storm _x)
    (rain _x)
    (< (windforce _x) 6) )
rfi-l> ;PROLOG Style analogously
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l>
rfi-l> ;uc rules
rfi-l> az (uc (cold _x) (no_sun _x) 0.3)
rfi-l> az (uc (bad_weather _x) (storm _x) 0.65)
rfi-l> az (uc (bad_weather _x) (cold _x) 0.25)
rfi-l>
rfi-l>
rfi-l> ;PROLOG Style
rfi-l> sp
rfi-p> l bad_weather
bad_weather(X) :-# storm(X), 0.65.
bad_weather(X) :-# cold(X), 0.25.
rfi-p> pause()
rfi-p>  bye
true
rfi-p>
rfi-p> sl
rfi-l> ;Translate uc clauses into footed clauses with a given uc factor
rfi-l> ;and combination rule.
rfi-l> uncertain *
rfi-l>
rfi-l>
rfi-l> l bad_weather
(ft (bad_weather _x)
    (* (storm _x) 0.65) )
(ft (bad_weather _x)
```

```
    (* (cold _x) 0.25) )
rfi-l>
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l>
rfi-l> ;PROLOG Style
rfi-l> sp
rfi-p> l bad_weather
bad_weather(X) :-& *(storm(X), 0.65).
bad_weather(X) :-& *(cold(X), 0.25).
rfi-p> pause()
rfi-p>  bye
true
rfi-p>
rfi-p> sl
rfi-l> ;***************************************************************
rfi-l> ;Becauce every premise has to return a uc value, the hornish
rfi-l> ;clauses have to be translated into footed clauses.
rfi-l> ;We declare all hn clauses as certain (uc factor = 1)
rfi-l>
rfi-l> footer 1
rfi-l> ;************************QUERY*********************************
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l>
rfi-l> (snow _x)
1
(_x = 12.10.92)
rfi-l> m
1
(_x = 12.24.92)
rfi-l>
rfi-l> m
unknown
```

```
rfi-l>
rfi-l> ;same query with tupof
rfi-l> (tupof (is _uc (snow _x)) '(tup _x _uc))
'(tup (tup 12.10.92 1) (tup 12.24.92 1))
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> (tupof (is _uc (no_sun _x)) '(tup _x _uc))
'(tup (tup 10.10.92 1) (tup 6.10.92 0.4))
rfi-l>
rfi-l>
rfi-l>
rfi-l> l cold
(ft (cold _x)
    (snow _x)
    1 )
(ft (cold _x)
    (* (no_sun _x) 0.3) )
rfi-l>
rfi-l>
rfi-l>
rfi-l> (tupof (is _uc (cold _x)) '(tup _x _uc))
'(tup (tup 12.10.92 1) (tup 12.24.92 1) (tup 10.10.92 0.3) (tup 6.10.92 0.12))
rfi-l>
rfi-l>
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l>
rfi-l> ;************************************************************
rfi-l> ;************************************************************
rfi-l> ;
rfi-l> ;In German law the pet owner is not always  responsible
rfi-l> ;for the damage caused by his pet. Responsible can be the owner,
rfi-l> ;the horseman, the buyer ...
rfi-l> ;In case of a pet causes a damage a judge has to decide who is
```

```
rfi-l> ;the 'pet holder'. The pet holder is described by a few attributes.
rfi-l> ;
rfi-l> ;                           pet holder
rfi-l> ;                          /    E     \
rfi-l> ;                         /            \
rfi-l> ;                        /              \
rfi-l> ;                       /                \
rfi-l> ;                   interest              master
rfi-l> ;                    / C  \              /  D   \
rfi-l> ;                   /      \            /        \
rfi-l> ;            immediate non straight possession authority
rfi-l> ;             interest    interest
rfi-l> ;            /   A  \     /  B  \
rfi-l> ;           /        |   |       \
rfi-l> ;          /         |   |        \
rfi-l> ;    costs of|advantage| profit | loss
rfi-l> ; maintenance| of use  | chances| chances
rfi-l> ;     -1-       -2-      -3-       -4-   -5-     -6-
rfi-l> ;
rfi-l> ;This attributs are measured in terms like:
rfi-l> ;
rfi-l> ; -1- shelter, food, nurture, doctor etc.
rfi-l> ; -2- receipts of work, hiring etc.
rfi-l> ; -3- costs of procuring
rfi-l> ; -4- initial costs, costs of education etc.
rfi-l> ; -5- actual power, use, shelter, nuture in the own household or
rfi-l> ;     farm...
rfi-l> ; -6- decision about the existance of the pet (life and use )
rfi-l> ;
rfi-l> ;From experience it is known that the combination rules in this
rfi-l> ;example differ:
rfi-l> ; A : max                      'normal or'
rfi-l> ; B : square root              'and with compensation'
rfi-l> ; C : cubical root with 1:2    'or with compensation 1:2'
rfi-l> ; D :  max                     'normal or'
rfi-l> ; E : cubical root with 2:1    'or with compensation 2:1'
rfi-l> ;
rfi-l> ;
rfi-l> ;For more details see Jan Heithecker in KI 3/1993.
```

```
rfi-l> ;****************************************************************
rfi-l> ;****************************************************************
rfi-l> (pause)
rfi-l>  bye
true
rfi-l> destroy
rfi-l>
rfi-l> az (uc (pet_hol _x) (int _x)(res _x) 1)
rfi-l>
rfi-l> ;combination rule 'cubical root with 2:1'
rfi-l> azft (cubroot2 _x _y 1) (expt (* _x _x _y) 1/3)
rfi-l>
rfi-l> uncertain cubroot2
rfi-l>
rfi-l> l
(ft (pet_hol _x)
    (cubroot2 (int _x) (res _x) 1) )
(ft (cubroot2 _x _y 1)
    (expt (* _x _x _y) 1/3) )
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;PROLOG style
rfi-l> sp
rfi-p> l
pet_hol(X) :-& cubroot2(int(X), res(X), 1).
cubroot2(X, Y, 1) :-& expt(*(X, X, Y), 1/3).
rfi-p> pause()
rfi-p>  bye
true
rfi-p> sl
rfi-l>
rfi-l> az (uc (int _x) (im_int _x) (ns_int _x) 1)
rfi-l>
rfi-l> ;combination rule 'cubical root with 1:2'
rfi-l> azft (cubroot1 _x _y 1) (expt (* _x _y _y) 1/3)
rfi-l>
```

```
rfi-l> uncertain cubroot1
rfi-l>
rfi-l> l int
(ft (int _x)
    (cubroot1 (im_int _x) (ns_int _x) 1) )
rfi-l>
rfi-l> (pause)

rfi-l>  bye
true
rfi-l>
rfi-l> ;PROLOG style
rfi-l> sp
rfi-p> l int
int(X) :-& cubroot1(im_int(X), ns_int(X), 1).
rfi-p> pause()
rfi-p>  bye
true
rfi-p> sl
rfi-l>
rfi-l>
rfi-l>
rfi-l> az (uc (ns_int _x) (prof _x ) (loss _x ) 1)
rfi-l>
rfi-l> ; square root
rfi-l> azft (sq _x _y 1) (sqrt (* _x _y))
rfi-l>
rfi-l> uncertain sq
rfi-l>
rfi-l> l ns_int
(ft (ns_int _x)
    (sq (prof _x) (loss _x) 1) )
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;PROLOG style
rfi-l> sp
```

```
rfi-p> l ns_int
ns_int(X) :-& sq(prof(X), loss(X), 1).
rfi-p> pause()
rfi-p>  bye
true
rfi-p> sl
rfi-l>
rfi-l>
rfi-l> az (uc (im_int _x) (cos_main _x) (use_adv _x) 0)
rfi-l> az (uc (res _x) (poss _x) (auth _x) 0)
rfi-l>
rfi-l> uncertain max
rfi-l>
rfi-l> l im_int
(ft (im_int _x)
    (max (cos_main _x) (use_adv _x) 0) )
rfi-l> l res
(ft (res _x)
    (max (poss _x) (auth _x) 0) )
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;PROLOG style
rfi-l> sp
rfi-p> l im_int
im_int(X) :-& max(cos_main(X), use_adv(X), 0).
rfi-p> l res
res(X) :-& max(poss(X), auth(X), 0).
rfi-p> pause()
rfi-p>  bye
true
rfi-p> sl
rfi-l>
rfi-l> ;*****************************************************************
rfi-l> ;*****************************************************************
rfi-l> ;These attributes are given in natural language. So they have to
rfi-l> ;be translated into numerical values.
```

```
rfi-l> ;*********************************************************************
rfi-l> ;*********************************************************************
rfi-l>
rfi-l> az (uc (nl null) 0.01)
rfi-l> az (uc (nl very_few) 0.1)
rfi-l> az (uc (nl few) 0.25)
rfi-l> az (uc (nl medium) 0.5)
rfi-l> az (uc (nl high) 0.75)
rfi-l> az (uc (nl very_high) 0.9)
rfi-l> az (uc (nl total) 1)
rfi-l>
rfi-l> az (ft (cos_main john) (nl very_few))
rfi-l> az (ft (use_adv  john) (nl  few))
rfi-l> az (ft (prof     john) (nl few))
rfi-l> az (ft (loss     john) (nl null))
rfi-l> az (ft (poss     john)  (nl total))
rfi-l> az (ft (auth     john) (nl medium))
rfi-l>
rfi-l> az (ft (cos_main mary)(nl total))
rfi-l> az (ft (use_adv  mary)(nl null))
rfi-l> az (ft (prof     mary) (nl few))
rfi-l> az (ft (loss     mary) (nl high))
rfi-l> az (ft (poss     mary) (nl null))
rfi-l> az (ft (auth     mary) (nl high))
rfi-l>
rfi-l> az (ft (cos_main bob) (nl total))
rfi-l> az (ft (use_adv  bob) (nl few))
rfi-l> az (ft (prof     bob) (nl total))
rfi-l> az (ft (loss     bob) (nl total))
rfi-l> az (ft (poss     bob) (nl null))
rfi-l> az (ft (auth     bob) (nl high))
rfi-l>
rfi-l> ;Translate hornish facts into certain footed facts (certain =1)
rfi-l> footer 1
rfi-l>
rfi-l> ;Translate uncertain facts into footed facts with a given
rfi-l> ;uc factor. The rule given to uncertain is at this point of
rfi-l> ;no importance, because facts only change their tags.
rfi-l>
```

```
rfi-l> uncertain *
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> l
(ft (pet_hol _x)
    (cubroot2 (int _x) (res _x) 1) )
(ft (cubroot2 _x _y 1)
    (expt (* _x _x _y) 1/3) )
(ft (int _x)
    (cubroot1 (im_int _x)(ns_int _x) 1) )
(ft (cubroot1 _x _y 1)
    (expt (* _x _y _y) 1/3) )
(ft (ns_int _x)
    (sq (prof _x) (loss _x) 1) )
(ft (sq _x _y 1)
    (sqrt (* _x _y)) )
(ft (im_int _x)
    (max (cos_main _x) (use_adv _x) 0) )
(ft (res _x)
    (max (poss _x) (auth _x) 0) )
(ft (nl null)
    0.01 )
(ft (nl very_few)
    0.1 )
(ft (nl few)
    0.25 )
(ft (nl medium)
    0.5 )
(ft (nl high)
    0.75 )
(ft (nl very_high)
    0.9 )
(ft (nl total)
    1 )
(ft (cos_main john)
    (nl very_few) )
(ft (use_adv john)
```

```
    (nl few) )
(ft (prof john)
    (nl few) )
(ft (loss john)
    (nl null) )
(ft (poss john)
    (nl total) )
(ft (auth john)
    (nl medium) )
(ft (cos_main mary)
    (nl total) )
(ft (use_adv mary)
    (nl null) )
(ft (prof mary)
    (nl few) )
(ft (loss mary)
    (nl high) )
(ft (poss mary)
    (nl null) )
(ft (auth mary)
    (nl high) )
(ft (cos_main bob)
    (nl total) )
(ft (use_adv bob)
    (nl few) )
(ft (prof bob)
    (nl total) )
(ft (loss bob)
    (nl total) )
(ft (poss bob)
    (nl null) )
(ft (auth bob)
    (nl high) )
rfi-l> ;PROLOG style analogously
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l>
```

```
rfi-l>
rfi-l> ;The query for 'Is john a pet holder'
rfi-l> (pet_hol john)
0.19407667236782145
rfi-l>
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;The query for all pet holders
rfi-l> (pet_hol _x)
0.19407667236782145
(_x = john)
rfi-l>
rfi-l> m
0.6263231749593858
(_x = mary)
rfi-l>
0.9085602964160698
(_x = bob)
rfi-l> m
unknown
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l>
rfi-l>
rfi-l> ;*****************APPLICATIONS********************************
rfi-l>
rfi-l> ;****************  >  ****************************************
rfi-l>
rfi-l> ;Now we can make a useful definition of pet holder, if the
rfi-l> ;uc factor of pet_hol is greater or equal to 0.5
rfi-l> (>= (pet_hol _x) 0.5)
false
(_x = john)
```

```
rfi-l>
rfi-l> m
true
(_x = mary)
rfi-l>
rfi-l> m
true
(_x = bob)
rfi-l>
rfi-l>
rfi-l> ;Enumerate all pet holders
rfi-l> (tupof (>= (pet_hol _x) 0.5) _x)
'(tup mary bob)
rfi-l>
unknown
rfi-l> ;Enumerate all pet holders with uc factors
rfi-l> (tupof (is _val (pet_hol _x)) (>= _val 0.5) '(tup _x _val))
'(tup (tup mary 0.6263231749593858) (tup bob 0.9085602964160698))
rfi-l>
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;****************reduce****************************************
rfi-l> ;The function reduce expects two arguments, where the first is a
rfi-l> ;function and the second is a list. It applies fct to all
rfi-l> ;elements of the list.
rfi-l> ;     (tup a b c) --> (fct a b c)
rfi-l>
rfi-l> az (ft (reduce _fct (tup _x)) _x)
rfi-l> az (ft (reduce _fct(tup _x _y | _rest))
          (_fct _x (reduce _fct '(tup _y | _rest))))
rfi-l>
rfi-l> ;The function maxp returns, given two tups, the tup with the
rfi-l> ;greater second argument (uc factor)
rfi-l> az (ft (maxp (tup _c1 _uc1) (tup _c2 _uc2))
                 (>= _uc1 _uc2) '(tup _c1 _uc1))
rfi-l> az (ft (maxp (tup _c1 _uc1) (tup _c2 _uc2))
```

```
                     (< _uc1 _uc2) '(tup _c2 _uc2))
rfi-l>
rfi-l> ;Query for pet holder with the greatest uc factor
rfi-l> (reduce maxp  (tupof (is _uc (pet_hol _x)) '(tup _x _uc)))
'(tup bob 0.9085602964160698)
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;***********************fetch*************************************
rfi-l> ;* * * * * * * * after an idea of Rick LeFaivre * * * * * * * * *
rfi-l> ;* * * * * * * * see 'Fuzzy Reference Manual'* * * * * * * * * *
rfi-l> ;
rfi-l> ;Fetch is called with one, three or four variabels. The required
rfi-l> ;first argument is the name of the predicate of which fetch
rfi-l> ;returns an assertion whose uncertainty factor is in the proper
rfi-l> ;range which is given through the last two arguments. It fails if
rfi-l> ;no assertion is found.If more than one solution is found, it
rfi-l> ;returns the one whose uncertainty value is closest to the lower
rfi-l> ;bound. The default range is [1,0]. Notice that you receive the
rfi-l> ;highest uncertainty value by specifying the range as
rfi-l> ;[upper, lower] and the lowest uncertainty value by
rfi-l> ;specifying it as [lower, upper]. For that you fetch the maximum
rfi-l> ;with the range [1, 0] and the minimum by [0, 1].
rfi-l> ;The second argument is the argument to which the predicate is
rfi-l> ;applied. It gets interesting if there is more than one solution
rfi-l> ;for such a query. Also here you can fetch the 'surest' or
rfi-l> ;'unsurest' solution.
rfi-l> ;The output of fetch is a list. The first element of this list
rfi-l> ;is the difference between the left bound and the uncertainty factor,
rfi-l> ;next is the uncertainty factor and last there is the binding.
rfi-l>
rfi-l>
rfi-l> az (ft (fetch _pred _from _to) (<= _from _to)
          (reduce minp (tupof  (is _val (_pred _x))
                               (<= _val _to)
                               (>= _val _from)
                               (is _diff (- _val _from))
```

```
                                '(tup _diff _val _x))))
unknown
rfi-l>
rfi-l> az (ft (fetch _pred _from _to) (> _from _to)
          (reduce minp (tupof  (is _val (_pred _x))
                                (>= _val _to)
                                (<= _val _from)
                                (is _diff (- _from _val))
                                '(tup  _diff _val _x))))
rfi-l>
rfi-l> az (ft (fetch _pred) (fetch _pred 1 0))
rfi-l>
rfi-l> az (ft (fetch _pred _predarg  _from _to) (<= _from _to)
          (reduce minp (tupof (is _val (_pred _predarg))
                               (<= _val _to)
                               (>= _val _from)
                                (is _diff (- _val _from))
                               '(tup _diff _val _predarg))))
rfi-l>
rfi-l> az (ft (fetch _pred _predarg _from _to) (> _from _to)
          (reduce minp (tupof (is _val (_pred _predarg))
                               (>= _val _to)
                               (<= _val _from)
                               (is _diff (- _from _val))
                               '(tup _diff _val _predarg))))
rfi-l>
rfi-l> az (ft (fetch _pred _predarg) (fetch _pred _predarg 1 0))
rfi-l>
rfi-l> az (ft (minp (tup _diff1 _val1 _x1) (tup _diff2 _val2 _x2))
                   (<= _diff1 _diff2) '(tup _diff1 _val1 _x1))
rfi-l> az (ft (minp (tup _diff1 _val1 _x1) (tup _diff2 _val2 _x2))
                    (> _diff1 _diff2) '(tup _diff2 _val2 _x2))
rfi-l>
rfi-l>
rfi-l>
rfi-l>
rfi-l> (fetch  pet_hol 0 1)
unknown
rfi-l>
```

```
rfi-l>
rfi-l>
rfi-l> (fetch pet_hol 1 0)
'(tup 0.09143970358393017 0.9085602964160698 bob)
rfi-l>
rfi-l>
rfi-l>
rfi-l> (fetch pet_hol)
'(tup 0.09143970358393017 0.9085602964160698 bob)
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;If there is more than one solution, one can fetch the one with
rfi-l> ;the greatest (smallest) uc factor.
rfi-l> ;Therefore we need another clause for mary:
rfi-l> az (ft (cos_main mary)(nl few))
rfi-l>
rfi-l> (fetch pet_hol mary 0 1)
'(tup 0.46026438677468706 0.46026438677468706 mary)
rfi-l>
rfi-l>
rfi-l>
rfi-l> (fetch pet_hol mary 1 0)
'(tup 0.37367682504061417 0.6263231749593858 mary)
rfi-l>
rfi-l>
rfi-l> (fetch pet_hol mary )
'(tup 0.37367682504061417 0.6263231749593858 mary)
rfi-l>
rfi-l>
rfi-l>
rfi-l> ;It is also possible to fetch a pet holder with a special
rfi-l> ;uncertainty value.
rfi-l> (fetch  pet_hol 1 1)
unknown
rfi-l> ;means that there is no known 100% pet holder.
rfi-l>
```

```
rfi-l>
rfi-l> (fetch pet_hol  0.9085602964160698  0.9085602964160698)
bob
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l> ;*****************quick sort***************************************
rfi-l> ;quick sort with an higher order function. It removes all double
rfi-l> ;answers.
rfi-l> az (ft ((qsort _cr) (tup)) '(tup) )
rfi-l> az (ft ((qsort _cr) (tup _x | _y))
       ('(partition _cr) _x _y _sm _gr)
       (appfun ('(qsort _cr) _sm)
               (tup _x | ('(qsort _cr) _gr)) ) )
rfi-l>
rfi-l>
rfi-l> az (hn ((partition _cr) _x (tup) (tup) (tup)))
rfi-l> az (hn ((partition _cr) _x (tup _x | _z) _l1 _l2 )
                 ('(partition _cr) _x _z _l1 _l2) )
rfi-l> az (hn ((partition _cr) _x (tup _y | _z) (tup _y | _sm) _gr )
                           (_cr _y _x)
                           ('(partition _cr) _x _z _sm _gr) )
rfi-l> az (hn ((partition _cr) _x (tup _y | _z) _sm (tup _y | _gr) )
                           (_cr _x _y)
                           ('(partition _cr) _x _z _sm _gr) )
rfi-l>
rfi-l> az (ft (appfun (tup) _l) _l )
rfi-l> az (ft (appfun (tup _h | _r) _l) (tup _h | (appfun _r _l)) )
rfi-l>
rfi-l>
rfi-l>
rfi-l> ;function secnd<= and secnd>=
rfi-l> az (hn (secnd<= ( id _m) ( id _n)) (<= _m _n) )
rfi-l> az (hn (secnd>= ( id _m) ( id _n)) (>= _m _n) )
rfi-l>
rfi-l>
rfi-l> ('(qsort secnd<=) (tupof (is _uc (pet_hol _x)) '(_x _uc)))
```

```
'(tup
  (john 0.19407667236782145)
  (mary 0.46026438677468706)
  (mary 0.6263231749593858)
  (bob 0.9085602964160698) )
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l>
rfi-l> ;**********************higher order combination rule *************
rfi-l> ;It is also possible to give the combination rule as a higher
rfi-l> ;order function.
rfi-l>
rfi-l> az (uc (cos_main _x) (shelter _x) (food _x)(nurture _x)(doctor _x) 0.8)
rfi-l>
rfi-l> uncertain  (cr)
rfi-l>
rfi-l> l cos_main
(ft (cos_main john)
    (nl very_few) )
(ft (cos_main mary)
    (nl total) )
(ft (cos_main bob)
    (nl total) )
(ft (cos_main mary)
    (nl few) )
(ft (cos_main _x)
    ((cr)
     (shelter _x)
     (food _x)
     (nurture _x)
     (doctor _x)
     0.8 ) )
rfi-l>
rfi-l> azft (cr) max
rfi-l>
rfi-l> azft (shelter tom) (nl high)
rfi-l> azft (food tom)(nl few)
```

```
rfi-l> azft (nurture tom) (nl very_high)
rfi-l> azft (doctor tom) (nl very_few)
rfi-l>
rfi-l> (cos_main tom)
0.9
rfi-l>
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
rfi-l>
rfi-l> ;change the higher order function
rfi-l>
rfi-l> rxft (cr) max
rfi-l> azft (cr)  *
rfi-l>
rfi-l> (cos_main tom)
0.013500000000000002
rfi-l>
rfi-l>
rfi-l> (pause)
rfi-l>  bye
true
rfi-l>
```

B The Uncertainty Translator

```
;test for uncertainty facts  ((uc (c...)  UC FACTOR))
(defun ucfact-t (clause)
        (and (uc-tt clause) (equal (length clause) 3)))


;test for uncertainty rule ((uc (c...) (ucfb1...) .. (ucfbM ...)
;UC FACTOR)
;The permises of the uncertainty clauses are functional and return
;an uncertainty factor in the interval [0 .. 1].
(defun uc-t (x) (and (consp x) (eq 'uc (car x))))
(defun uc-tt (x) (and (uc-t x) (< 2 (length x))))

;Neither the compiler nor the interpreter can use uc clauses, so they have
;to be translated into "normal" footed clauses (ft clauses)
(defun replace-first (clause old new)
     (and (eq (car clause) old) (cons new (cdr clause))))

(defun mk-kombfct (body rule)
          (cons rule body))

(defun from-uc-to-ft (clause rule)
  (cond ((ucfact-t clause) (replace-first clause 'uc 'ft))
        (  (uc-tt clause)  (list 'ft (s-head clause)
                                     (mk-kombfct (s-premises clause) rule )))))

;Translate a database (hn & ft clauses stay unmodified)
(defun uncertain-clause  (clause rule)
  (cond ((uc-tt clause) (from-uc-to-ft clause rule ))
        (t clause)))

(defun uncertain-database (db rule)
    (mapcar #'(lambda (clause) (uncertain-clause clause rule))  db ))

;Translate and save the *rfi-database* (RELFUN database)
(defun uncertain-db ( rule)
       (setq  *rfi-database* (uncertain-database *rfi-database* rule )) 'true)
```

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter 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 via anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.

The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-93-14
Joachim Niehren, Andreas Podelski, Ralf Treinen:
Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support
Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-17
Rolf Backofen:
Regular Path Expressions in Feature Logic
37 pages

RR-93-18
Klaus Schild: Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard Hollunder:
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller:
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy:
Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel:
Document Highlighting —
Message Classification in Printed Business Letters
17 pages

RR-93-25
Klaus Fischer, Norbert Kuhn: A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-26
Jörg P. Müller, Markus Pischel: The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-27
Hans-Ulrich Krieger:
Derivation Without Lexical Rules
33 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker: Feature-Based Allomorphy
8 pages

RR-93-29
Armin Laux: Representing Belief in Multi-Agent Worlds viaTerminological Logics
35 pages

RR-93-30
Stephen P. Spackman, Elizabeth A. Hinkelman:
Corporate Agents
14 pages

RR-93-31
Elizabeth A. Hinkelman, Stephen P. Spackman:
Abductive Speech Act Recognition, Corporate Agents and the COSMA System
34 pages

RR-93-32
David R. Traum, Elizabeth A. Hinkelman:
Conversation Acts in Task-Oriented Spoken Dialogue
28 pages

RR-93-33
Bernhard Nebel, Jana Koehler:
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.):
Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt: Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-38
Stephan Baumann: Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf:
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-41
Winfried H. Graf: LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-42
Hubert Comon, Ralf Treinen:
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-43
M. Bauer, G. Paul: Logic-based Plan Recognition for Intelligent Help Systems
15 pages

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt: Subsumption between Queries to Object-Oriented Databases
36 pages

RR-93-45
Rainer Hoch: On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

RR-93-46
Philipp Hanschke: A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

RR-93-48
Franz Baader, Martin Buchheit, Bernhard Hollunder:
Cardinality Restrictions on Concepts
20 pages

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

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

RR-94-03
Gert Smolka:
A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 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-94-06
Dietmar Dengler:
An Adaptive Deductive Planning System
17 pages

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

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

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

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

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

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

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Towards a Sharable Knowledge Base on Recyclable Plastics
14 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-94-16
Gert Smolka: A Foundation for Higher-order Concurrent Constraint Programming
26 pages

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

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

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz:
Encapsulated Search and Constraint Programming in Oz
21 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-94-33
Franz Baader, Armin Laux:
Terminological Logics with Modal Operators
29 pages

DFKI Technical Memos

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-04
Hans-Günther Hein:
Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

TM-93-05
Michael Sintek: Indexing PROLOG Procedures into DAGs by Heuristic Classification
64 pages

TM-94-01
Rainer Bleisinger, Klaus-Peter Gores:
Text Skimming as a Part in Paper Document Understanding
14 pages

TM-94-02
Rainer Bleisinger, Berthold Kröll:
Representation of Non-Convex Time Intervals and Propagation of Non-Convex Relations
11 pages

DFKI Documents

D-93-14
Manfred Meyer (Ed.): Constraint Processing – Proceedings of the International Workshop at CSAM'93, July 20-21, 1993
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-15
Robert Laux:
Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-16
Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Gabriele Schmidt: Design & KI
74 Seiten

D-93-20
Bernhard Herbig:
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-21
Dennis Drollinger:
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten

D-93-22
Andreas Abecker:
Implementierung graphischer Benutzungsoberflächen mit Tcl/Tk und Common Lisp
44 Seiten

D-93-24
Brigitte Krenn, Martin Volk:
DiTo-Datenbank: Datendokumentation zu Funktionsverbgefügen und Relativsätzen
66 Seiten

D-93-25
Hans-Jürgen Bürckert, Werner Nutt (Eds.):
Modeling Epistemic Propositions
118 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-26
Frank Peters: Unterstützung des Experten bei der Formalisierung von Textwissen
INFOCOM:
Eine interaktive Formalisierungskomponente
58 Seiten

D-93-27
Rolf Backofen, Hans-Ulrich Krieger, Stephen P. Spackman, Hans Uszkoreit (Eds.):
Report of theEAGLES Workshop on Implemented Formalisms at DFKI, Saarbrücken
110 pages

D-94-01
Josua Boon (Ed.):
DFKI-Publications: The First Four Years
1990 - 1993
75 pages

D-94-02
Markus Steffens: Wissenserhebung und Analyse zum Entwicklungsprozeß eines Druckbehälters aus Faserverbundstoff
90 pages

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

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

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

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

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

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

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 only for a nominal charge of 25 DM (or 15 US-$).

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 Seiten

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

Uncertainty-Valued Horn Clauses

Victoria Hall

TM-94-03

Technical Memo