Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Technical Memo
TM-91-04

A Sampler of Relational/Functional Definitions

Harold Boley (Ed.)

March 1991

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, 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 und Saarbrücken is a non-profit organization which was founded in 1988 by the shareholder companies ADV/Orga, AEG, IBM, Insiders, Fraunhofer Gesellschaft, GMD, Krupp-Atlas, Mannesmann-Kienzle, Nixdorf, Philips 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
- Intelligent Communication Networks
- Intelligent Cooperative Systems.

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.

Prof. Dr. Gerhard Barth
Director

A Sampler of Relational/Functional Definitions

Harold Boley (Ed.)

DFKI-TM-91-04

A Sampler of Relational/Functional Definitions

Harold Boley (Ed.)

Abstract

This is a collection of annotated RELFUN definitions showing principles and applications of relational/functional specification. It consists of concise declarative programs (often invertible) selected on the basis of didactic considerations. The knowledge they encode is mostly derived from the domain of mechanical engineering. The definitions solve problems in solid geometry, feature parsing, workpiece normalization, chemistry, etc. All examples can be run directly in RELFUN.

Contents

```
;     A brief introduction to basic RELFUN, using its LISP-like syntax
;                              Harold Boley                              Mar 1991

; In RELFUN, a LISP list (e1 ... eN) or PROLOG list [e1,...,eN] is represented
; as an (N-)tup(le) structure (tup e1 ... eN). Logical variables are marked by
; an underscore, "_". The PROLOG list pattern [1,2|V] becomes (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 'hornish'
; (or hn) structure (hn (c ...) (b1 ...) .. (bM ...)), where facts just have no
; premise (M=0). A conditional equation g(...) = f(...) if b1(...), .., bM(...)
; in RELFUN is generalized to (ft (g ...) (b1 ...) .. (bM ...) (f ...)), an ft
; (or 'footed') clause, whose "b" premises may accumulate partial results and
; whose "f" premise returns the (non-)ground/(non-)deterministic values.

; As in LISP, nestings like (+ (* 3 3) (1- 8)) => 16 are reduced call-by-value.
; "Passive" structures are marked by a backquote, "`", except in clause heads.
; RELFUN's tup FUNCTION returns the tup STRUCTURE of its evaluated arguments:
  (ft (tup | _r) `(tup | _r))  ; _r will be bound to the tup of all arguments.
; This permits calls like (is _t 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 relational style (inversion (revrel _u (tup 1 2)) loops on MORE):

(hn (apprel (tup) _l _l))
(hn (apprel (tup _h | _r) _m (tup _h | _s))  (apprel _r _m _s))
(hn (revrel (tup) (tup)))
(hn (revrel (tup _h | _r) _l) (revrel _r _m) (apprel _m `(tup _h) _l))

; LISP-like functional style (inversion (is (tup 1 2) (revfun _u)) not needed):

(ft (appfun (tup) _l) _l)
(ft (appfun (tup _h | _r) _l) (tup _h | (appfun _r _l))) ;NESTED (user) form ->
;(ft (appfun (tup _h | _r) _l) (is _1 (appfun _r _l)) (tup _h | _1)) ;FLATTENED
(ft (revfun (tup)) `(tup))
(ft (revfun (tup _h | _r)) (appfun (revfun _r) `(tup _h))) ;nest and (compiler)
;(ft (revfun (tup _h | _r)) (is _1 (revfun _r)) (appfun _1 `(tup _h))) ;flatten

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

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

```
;           Symbolic and numeric RELFUN factorial fixpoints
;                          Harold Boley                              Mar 1991

; Factorial can be defined numerically or symbolically, both in the relational
; and functional RELFUN subsets. The symbolic versions can be inverted in both
; formulations, as shown by fixpoint computations (no cuts are used).

; Relational factorial for symbolic and numeric use

(hn  (facrel 0  1)     (num))  ; numeric mode iff (num) is asserted
(hn  (facrel 0  (s 0)))        ; symbolic mode with integers as s(uccessor) terms
(hn  (facrel _n _r)  (sub1rel _n _p) (facrel _p _v) (multrel _n _v _r))

; Functional factorial for symbolic and numeric use

(ft  (facfun 0)    (num) 1)  ; numeric mode iff (num) is asserted
(ft  (facfun 0)    `(s 0))   ; symbolic mode with integers as s(uccessor) terms
(ft  (facfun _n)   (multfun _n (facfun (sub1fun _n))))

; Equivalent FLATTENed version of recursive clause, as produced by the compiler

; (ft  (facfun _n)  (is _2 (sub1fun _n)) (is _1 (facfun _2)) (multfun _n _1))

; Arithmetic operators ...

(hn  (sub1rel _n _p)      (is _p (sub1fun _n)))
(hn  (multrel _n _v _r)   (is _r (multfun _n _v)))

; ... implementable via numeric LISP builtins (non-invertible arithmetics)

(ft  (sub1fun _n)      (num) (1- _n))
(ft  (multfun _n _v)   (num) (* _n _v))

; ... or symbolically (invertible arithmetics)

(ft  (sub1fun (s _n))      _n)
(ft  (add1fun _n)          `(s _n))
(ft  (plusfun 0       _v)  _v)
(ft  (plusfun (s _n)  _v)  (plusfun _n (add1fun _v)))
(ft  (multfun 0       _v)  0)
(ft  (multfun (s _n)  _v)  (plusfun _v (multfun _n _v)))

; rfi> (facrel _x _x)        ; Relational fixpoint computation:
; true
; (_x = (s 0))               ; the "smallest"
; rfi> more
; true
; (_x = (s (s 0)))           ; the "largest" (another MORE would diverge)
; rfi> (is _x (facfun _x))   ; Functional fixpoint computation:
; (s 0)
; (_x = (s 0))               ; the "smallest"
; rfi> more
; (s (s 0))
; (_x = (s (s 0)))           ; the "largest" (another MORE would diverge)
; rfi> azhn (num)      ; Switching to numeric mode:
; rfi> (facrel 1 1)    ; (facrel _x _x) would now lead to "free variable" error
; true
; rfi> (facrel 2 _w)   ; (facrel _w 2) would here lead to "free variable" error
; true
; (_w = 2)
; rfi> (facfun 57)     ; The symbolic mode would not be suited for large integers
; 40526919504877216755680601905432322134980384796226602145184481280000000000000
```

```
;                Two invertible RELFUN sort functions
;                        Harold Boley                          Mar 1991

; These slowsort and quicksort versions "with duplicates" show a relational and
; several functional uses of RELFUN's non-ground calls.


;; A functional slowsort program

; Non-deterministic permutations are filtered through sorted:
(ft (ssort _x) (sorted (perm _x)))

; Return sorted lists unchanged, fail for unsorted ones:
(ft (sorted (tup)) `(tup))
(ft (sorted (tup _x)) `(tup _x))
(ft (sorted (tup _x _y | _z)) (lesseq _x _y) (tup _x | (sorted `(tup _y | _z))))

; Non-ground delete function call returns _u-less list and binds _u for tup use:
(ft (perm (tup)) `(tup))
(ft (perm (tup _x | _y)) (tup _u | (perm (delete _u `(tup _x | _y)))))

; Non-deterministic delete-element-from-list function:
(ft (delete _x (tup _x | _z)) _z)
(ft (delete _x (tup _y | _z)) (tup _y | (delete _x _z)))

; A less-or-equal relation over s-terms:
(ft (lesseq 0 _x) true)
(ft (lesseq (s _x) (s _y)) (lesseq _x _y))

; The call (tup  |  ) evaluates its arguments, unlike the structure `(tup  |  ):
(ft (tup | _r) `(tup | _r))

; The non-ground sample call (ssort `(tup (s (s 0)) _e (s 0))) returns:
; 1. (tup 0 (s 0) (s (s 0)))              and binds _e = 0
; 2. (tup (s 0) (s 0) (s (s 0)))          and binds _e = (s 0)
; 3. (tup (s 0) (s (s 0)) (s (s _x:32)))  and binds _e = (s (s _x:32))
; 4. (tup (s 0) (s 0) (s (s 0)))          and binds _e = (s 0)
; 5. (tup (s 0) (s (s 0)) (s (s 0)))      and binds _e = (s (s 0))
; Results 2. and 4. bind _e to (s 0) in different places of the sorted list


;; A (mostly) functional quicksort program

; Partition/recurse into lists of elements _sm(aller) and _gr(eater) than _x:
(ft (qsort (tup)) `(tup))
(ft (qsort (tup _x | _y)) (partition _x _y _sm _gr)
                          (appfun (qsort _sm) (tup _x | (qsort _gr))))

; Relational definition makes delivery of 2 outputs (_sm and _gr) easy:
(hn (partition _x (tup) (tup) (tup)))
(hn (partition _x (tup _y | _z) (tup _y | _sm) _gr) (lesseq _y _x)
                                                    (partition _x _z _sm _gr))
(hn (partition _x (tup _y | _z) _sm (tup _y | _gr)) (lesseq _x _y)
                                                    (partition _x _z _sm _gr))

; Functional append:
(ft (appfun (tup) _l) _l)
(ft (appfun (tup _h | _r) _l) (tup _h | (appfun _r _l)))

; The sample call (qsort `(tup (s (s 0)) _e (s 0))) gives the same 5 results as
; ssort did, albeit in different order (and time).
```

```
;                A relational/functional analogy finder
;                 H. Boley, P. Hanschke, R. Scheubrein          21 Feb 1991

; This program is adapted from Sterling, Shapiro "The Art of Prolog", page 230.
; "analogy" tries to find an analogy between two pairs of figures
;    1. find a rule named _rule to relate figure _a to figure _b
;    2. apply _rule to figure _c to generate new figure _x
;    3. see if _x is in the list of possible _answers
;    4. return result _x

;;; main procedures ------------------------------------------------
(ft (analogy _a _b _c _answers)
    (_rule _a is-to _b)                ; _rule is a free relational variable
    as                  ; redundant
    (_rule _c is-to _x)                ; _rule is bound to the relation found
    (member-r _x _answers)
    _x)

(hn (inside-out  (within _figure1 _figure2)
                 is-to
                 (within _figure2 _figure1)))
(hn (upside-down (above  _figure1 _figure2)
                 is-to
                 (above  _figure2 _figure1)))
(hn (inside-down (within _figure1 _figure2)
                 is-to
                 (above  _figure2 _figure1)))

(hn !(member-r _x (tup _x | _tail)))
(hn (member-r _x (tup _head | _tail))
    (member-r _x _tail))

;;; testing --------------------------------------------------------
(ft (test-analogy _name)
    (figures _name _a _b _c)               ; binds a,b,c
    (answers _name _answers)               ; binds answers
    (analogy _a _b _c _answers))           ; calls program

(ft (pattern-analogy _myfig _yourfig)      ; classical analogy problem
    (analogy `(above square _myfig)        ; a:b = c:x with variables
             `(above _myfig square)        ; IN a,b (_myfig), and c,x
             `(above circle _yourfig)      ; (_yourfig). The id permits
             id))                          ; arbitrary x answers

;;; data -----------------------------------------------------------
(hn (figures test1
          (within square triangle)         ; figure a
          (within triangle square)         ; figure b
          (within circle square)))         ; figure c

(hn (answers test1
       (tup (within circle triangle)       ; list of possible
            (within square circle)         ; answers
            (within triangle square))))

;;; sample calls ---------------------------------------------------
;
;    rfi> consult "analogy"
;
;    rfi> (test-analogy test1)
;    (within square circle)
;
;    rfi> (pattern-analogy _myfig _yourfig)
;    (above _figure2:4 circle)
;    (myfig = _figure2:3)
;    (yourfig = _figure2:4)
;
;    rfi> (analogy `(within jonas fish)  (above fish jonas) `(within martian ufo) id)
;    (above ufo martian)
```

```
;          Attribute-centered and object-centered presentation of
;                 solid-geometry knowledge in RELFUN
;                              Harold Boley                          Mar 1991

; Functions computing diagonals, areas, curfaces, and volumes of cubes, blocks,
; cylinders, cones, and truncones for symbolic (direct) or numeric ("@") use:

(ft (diagonal (cube _x)) `(* _x (sqrt 3)))
(ft (area (cube _x)) `(* 6 (expt _x 2)))
(ft (volume (cube _x)) `(expt _x 3))

(ft (diagonal (block _x _y _z)) `(pyth _x _y _z))
(ft (area (block _x _y _z)) `(* 2 (+ (* _x _y) (* _x _z) (* _y _z))))
(ft (volume (block _x _y _z)) `(* _x _y _z))

(ft (area (cylinder _r _h)) `(* 2 (pi) _r (+ _r _h)))
(ft (curface (cylinder _r _h)) `(* 2 (pi) _r _h))
(ft (volume (cylinder _r _h)) `(* (pi) (expt _r 2) _h))

(ft (area (cone _r _h)) `(* (pi) _r (+ _r (pyth _r _h))))
(ft (curface (cone _r _h)) `(* (pi) _r (pyth _r _h)))
(ft (volume (cone _r _h)) `(* (/ (pi) 3) (expt _r 2) _h))

(ft (area (truncone _r1 _r2 _h))
    `(+ @(curface `(truncone _r1 _r2 _h))
        (* (pi) (expt _r1 2))
        (* (pi) (expt _r2 2))))
(ft (curface (truncone _r1 _r2 _h)) `(* (pi) (pyth (- _r1 _r2) _h) (+ _r1 _r2)))
(ft (volume (truncone _r1 _r2 _h))
    `(* (/ (pi) 3) (+ (expt _r1 2) (* _r1 _r2) (expt _r2 2)) _h))

(ft (area (sphere _r)) `(* 4 (pi) (expt _r 2)))
(ft (volume (sphere _r)) `(* (/ 4 3) (pi) (expt _r 3)))

; Some auxiliary definitions

(ft (pi) 3.141592653589793) ; a parameterless function used as a global constant
(ft (pyth _a _b) (sqrt (+ (expt _a 2) (expt _b 2))))
(ft (pyth _a _b _c) (sqrt (+ (expt _a 2) (expt _b 2) (expt _c 2))))

; Sample listings and calls:

; rfi> l curface                          ; Attribute-centered listing,
; (ft (curface (cylinder _r _h))          ; comparable to a PROLOG procedure
;     `(* 2 (pi) _r _h) )                 ; curface/1 or, to a polymorphic ML
; (ft (curface (cone _r _h))              ; function, defined for cylinder, cone
;     `(* (pi) _r (pyth _r _h)) )         ; and truncone terms
; (ft (curface (truncone _r1 _r2 _h))
;     `(* (pi) (pyth (- _r1 _r2) _h) (+ _r1 _r2)) )

; rfi> l (_att (cylinder | _par))         ; Object-centered listing,
; (ft (area (cylinder _r _h))             ; comparable to a LISP property list
;     `(* 2 (pi) _r (+ _r _h)) )          ; [(cylinder _r _h)
; (ft (curface (cylinder _r _h))          ;   < area     `(* 2 (pi) _r (+ _r _h)) >
;     `(* 2 (pi) _r _h) )                 ;   < curface  `(* 2 (pi) _r _h) >
; (ft (volume (cylinder _r _h))           ;   < volume   `(* (pi) (expt _r 2) _h) >
;     `(* (pi) (expt _r 2) _h) )          ; of a PARAMETERIZED obj (cylinder _r _h)

; rfi> (area `(cylinder 3 8))             ; symbolic result obtained by direct call
; (* 2 (pi) 3 (+ 3 8))
; rfi> @(area `(cylinder 3 8))            ; numeric result obtained via "@" (ecal)
; 207.34512
```

```
;; Towards Functional/Relational Engineering Knowledge in RELFUN

;; Harold Boley       -       Kaiserslautern       -       October  1989
;;                                                  Revised: February 1991


; Deterministic functions

(ft (weight _volume _material) (* _volume (specific-gravity _material)))
; LISP: (defun weight (volume material) (* volume (specific-gravity material)))

(ft (specific-gravity steel)   7.8)
; LISP: (defun specific-gravity (stuff) (if (eq stuff 'steel) 7.8 'unknown))

(ft (composed-of steel)        iron)         ; steel is composed of iron ...


; Non-deterministic function

(ft (alloyed-with steel) carbon)             ; ... and alloyed with carbon etc.
(ft (alloyed-with steel) manganese)
(ft (alloyed-with steel) chromium)


; Relation: Non-deterministically invertible iff (lessp) = sym-lessp
; [A workpiece is (strongly) monotone if its disk-segment diameters increase]

(hn (monotone (workpiece)))
(hn (monotone (workpiece _disk)))
(hn (monotone (workpiece _disk1 _disk2 | _disks))
 ((lessp) _disk1 _disk2)
 (monotone (workpiece _disk2 | _disks)))


; Self-normalizing data structures (here just returning workpiece/disk terms)

(ft (workpiece | _disks) `(workpiece | _disks))
(ft (disk | _elements)   `(disk | _elements))


; Numeric (non-invertible) and symbolic (invertible) lessp definitions

(hn (num-lessp _diameter1 _diameter2) (< _diameter1 _diameter2))

(hn (sym-lessp  (disk)                    (disk id | _elements)))
(hn (sym-lessp  (disk id | _elements1) (disk id | _elements2))
 (sym-lessp  (disk | _elements1)    (disk | _elements2)))


; Switch for setting the lessp version

(ft (lessp) sym-lessp)


; Sample calls [try (a), ..., (e) with more, more, ...]

(ft (a)  (alloyed-with steel))
(ft (b)  (is manganese (alloyed-with _x)) _x)
(ft (c)  (monotone (workpiece (disk a) (disk 1 2 3) (disk k l m n o))))
(ft (d)  (monotone (workpiece _x (disk 1 2 3) _y))  `(tup _x _y))
(ft (e)  (monotone (workpiece (disk _u _v _u) _x (disk _u | _y)))
         `(tup _x _y _u _v))
```

```
;       Qualitative geometry and higher-order parsing in RELFUN
;                              Harold Boley                                Mar 1991

; Self-normalizing qualitative-workpiece (qwp) terms consist of horizontal (h)
; [oriented left to right], up (u) and down (d) arguments that represent lines:

(ft (qwp) `(qwp))  ; clauses are directed equations of qwp-constructor algebra:
(ft !(qwp _x _x | _y) (qwp _x | _y))  ; idempotence (QUALITATIVE h, u, d lines)
(ft !(qwp u d | _y) unknown)  ; up/down never adjacent ("infinitely thin wall")
(ft !(qwp d u | _y) unknown)  ; down/up never adjacent ("infinitely thin wall")
(ft (qwp _x | _y) (is (qwp | _n) (qwp | _y)) `(qwp _x | _n))

; Parser applying an operator parameter to N-splits of qwp (|qwp| >= N >= 2):

(ft (parse _o (qwp)) `(tup))  ; empty workpiece (qwp) yields empty tuple (tup)
(ft (parse _o (qwp _x | _y))  ; non-empty qwp (we'll bind _o to feature or pal):
    (apprel _pre `(tup _i _j | _post) `(tup _x | _y))  ; split qwp args as a tup
    (apprel _pre `(tup _i _j) _pij)   ; _pij (prefix plus infix) has length >= 2
    (tup (_o | _pij)  ; generate _o-values for _pij's elements or fail
         |  ; first parse _post elements only, on failure also reuse _j
         (or `(parse _o `(qwp | _post))       `(parse _o `(qwp _j | _post)))))

; Auxiliaries:
(ft (tup | _r) `(tup | _r))  ; _r will be bound to the tup of all arguments
(hn (apprel (tup) _l _l))    ; PROLOGish append relation
(hn (apprel (tup _h | _r) _m (tup _h | _s))  (apprel _r _m _s))
(ft (or _g | _y) @_g)        ; first try to "@"-call goal _g (input passively),
(ft (or _g | _y) (or | _y))  ; then try rest of goals _y: realizes PROLOG's ";"

; A (ft (feature t1 ... tN) f) is a lexicon rule f -> t1 ... tN that RETURNS f:

(ft (feature h u) rightshoulder)  ; Binary features for contours   |  and   |
(ft (feature d h) leftshoulder)   ;                          right:  -  left:   -
                                  ;
(ft (feature h d) rightnose)      ;                                  -        -
(ft (feature u h) leftnose)       ;                          right:   | left:  |

(ft (feature h d h) qmark)        ; Ternary features to describe  -   and    -
(ft (feature h u h) swing)        ;                                "?":   |  "s":  |
                                  ;                                    -       -
(ft (feature d h u) groove)       ;                                dhu: | |  uhd:   -
(ft (feature u h d) collar)       ;                                     -        | |

; Using these rules, (parse feature (qwp h h h)) fails, but most calls succeed:

                                  ;  -          When normalized this qwp has
(ft (tst) (qwp u h d d h u h h h d))  ; | |  ---   40 parsings, incl. the final
                                  ;    | |   |   feature (non-terminal) list:
                                  ;     -        (tup collar groove collar).
; rfi> (parse feature (qwp d h u))    ; This qwp is normalized via call-by-value
; (tup leftshoulder rightshoulder)    ; and then produces 2 parsings:
; rfi> more                           ; a list consisting of a leftshoulder and
; (tup groove)                        ; a rightshoulder feature, and a unit
; rfi> more                           ; list containing their "h-overlapping"
; unknown                             ; aggregation, the groove feature.
; (parse feature `(qwp d _x u)), with a passive non-ground term, binds _x to h

; Instead of extensional feature rules use recursive pal operator as parameter:
(ft (pal) `(pal))     (ft (pal _c) `(pal _c))  ; palindrome argument sequences
(ft (pal _f | _r) (apprel _m `(tup _f) _r) (is _h (pal | _m)) `(pal _f _h _f))
; (parse pal (qwp h u h d h u h)) returns a list of "h-overlapping" pal parses,
; (tup (pal h (pal u) h) (pal h (pal d) h) (pal h (pal u) h)),  and a unit list.
```

```
; ARC-TEC Discussion Paper 90-03                                  9 May 1990
;                                                    Revised: 26 Mar 1991

;  A Self-Normalizing Term Representation of Rotation-Symmetric Workpieces

;                              Harold Boley


; A Workpiece is written as a term with the functor wp and a variable number
; of arguments representing the contour line above a symmetry axis called z.
; Each argument may be a curve term of the form
;        (curve-type begin-point end-point | further-properties)
; where curve-type is currently Line [l] or Circle-concAve [ca]. An argument
; may also be a point term (p z-coordinate x-coordinate), e.g. a degenerated
; line term that reduces to its single point.
; The RELFUN definitions first normalize the wp arguments in a call-by-value
; fashion and then check them for `well-formedness': consecutive curve terms
; must share a point, isolated points must coincide with adjacents points or
; curve points, etc.; the reduced wp term is returned. Ill-formed terms lead
; to failures [unknown].
; It is not checked whether the contour line somewhere cuts itself. Thus, wp
; acts as a (mostly) self-normalizing constructor for workpiece terms.
; Over such wp terms operations relevant for their NC-program production are
; defined (e.g. workpiece length and radius).


; The empty workpiece:
(ft !(wp)
      `(wp))

; Starting from the point (0,0):
(ft !(wp (_f (p 0 0) _end | _t) | _rem)
      (wph `(_f (p 0 0) _end | _t) | _rem))
(ft !(wp (p 0 0) | _rem)
      (wph `(p 0 0) | _rem))

; Ending at some point (z,0):
(ft !(wph (_f _begin (p _z 0) | _t))
      `(wp (_f _begin (p _z 0) | _t)))
(ft !(wph (p _z 0))
      `(wp))

; Checking adjacent curve terms (first's end = second's begin; no z-touch):
(ft !(wph (_f (p _z1 _x1) (p _z2 _x2) | _t)
          (_g (p _z2 _x2) (p _z3 _x3) | _u) | _rem)
      (> _x2 0)
      (is (wp | _s) (wph `(_g (p _z2 _x2) (p _z3 _x3) | _u) | _rem))
      `(wp (_f (p _z1 _x1) (p _z2 _x2) | _t) | _s))

; Joining identical adjacent points:
(ft !(wph (p _z _x) (p _z _x) | _rem)
      (wph `(p _z _x) | _rem))

; Joining a point with an adjacent curve term:
(ft !(wph (p _z _x) (_f (p _z _x) _end | _t) | _rem)
      (wph `(_f (p _z _x) _end | _t) | _rem))

; Joining a curve term with an adjacent point:
(ft !(wph (_f _begin (p _z _x) | _t) (p _z _x) | _rem)
      (wph `(_f _begin (p _z _x) | _t) | _rem))
```

```
; Constructing lines:
(ft !(l _begin _begin) _begin)
(ft (l _begin _end) `(l _begin _end))

; Constructing circle segments that are concave:
(ft (ca _begin _begin _center) _begin)
(ft (ca _begin _end _center)
     (= (distance _begin _center) (distance _end _center))
     `(ca _begin _end _center))

; Constructing points:
(ft (p _z _x) `(p _z _x))

; Selecting point coordinates:
(ft (s-z (p _z _x)) _z)
(ft (s-x (p _z _x)) _x)


; Elementary geometric primitive:
(ft (distance (p _z1 _x1) (p _z2 _x2))
     (sqrt (+ (expt (- _z1 _z2) 2) (expt (- _x1 _x2) 2))))


; Specific workpiece operations:

; Workpiece length:
(ft !(wplength (wp)) 0)
(ft !(wplength (wp (_f _begin (p _z 0) | _t))) _z)
(ft (wplength (wp (_f (p _z1 _x1) (p _z2 _x2) | _t)
                  (_g (p _z2 _x2) (p _z3 _x3) | _u) | _rem))
    (wplength `(wp (_g (p _z2 _x2) (p _z3 _x3) | _u) | _rem)))

; Workpiece curve-term types:
(ft !(wptypes (wp)) (tup))
(ft !(wptypes (wp (_f _begin (p _z 0) | _t))) (tup _f))
(ft (wptypes (wp (_f (p _z1 _x1) (p _z2 _x2) | _t)
                 (_g (p _z2 _x2) (p _z3 _x3) | _u) | _rem))
    (is (tup | _s) (wptypes `(wp (_g (p _z2 _x2) (p _z3 _x3) | _u) | _rem)))
    (tup _f | _s))

; Workpiece edges (contour points):
(ft !(wpedges (wp)) (tup))
(ft !(wpedges (wp (_f _begin (p _z 0) | _t))) (tup `(p _z 0)))
(ft (wpedges (wp (_f (p _z1 _x1) (p _z2 _x2) | _t)
                 (_g (p _z2 _x2) (p _z3 _x3) | _u) | _rem))
    (is (tup | _s) (wpedges `(wp (_g (p _z2 _x2) (p _z3 _x3) | _u) | _rem)))
    (tup `(p _z1 _x1) | _s))

; Workpiece edges' x-coordinates:
(ft (wpxes (wp | _terms)) (maptup s-x (wpedges `(wp | _terms))))

; Workpiece radius (maximum x-coordinate):
(ft (wpradius (wp | _terms))
    (is (tup | _s) (wpxes `(wp | _terms)))
    (max | _s))


; Tuples:
(ft (tup | _elems) `(tup | _elems))
(ft (constup _head (tup | _tail)) `(tup _head | _tail))
(ft (maptup _f (tup)) `(tup))
(ft (maptup _f (tup _head | _tail))
```

```
    (constup (_f _head) (maptup _f `(tup | _tail))))


; Two sample-workpiece normalization calls in a tup (equal when normalized):
(ft (test)   (tup  (wp (l (p 0 0) (p 0 12.5))     ; this wp is normal except _a
                       (l (p 0 12.5) (p 27 44.5))
                       (l (p 27 44.5) (p 44 44.5))
                       (l (p 44 44.5) (p _a 20))
                       (l (p 44 20) (p 149 20))
                       (l (p 149 20) (p 149 15))
                       (l (p 149 15) (p 183 15))
                       (ca (p 183 15) (p 185 13) (p 185 15))
                       (l (p 185 13) (p 185 0)))
                   (wp (p 0 0)                    ; this wp needs normalization
                       (l (p 0 0) (p 0 12.5))
                       (l (p 0 12.5) (p 0 12.5))
                       (l (p 0 12.5) (p 27 44.5))
                       (p 27 44.5)
                       (l (p 27 44.5) (p 44 44.5))
                       (p 44 44.5)
                       (p 44 44.5)
                       (p 44 44.5)
                       (l (p 44 44.5) (p _a 20))
                       (l (p 44 20) (p 149 20))
                       (l (p 149 20) (p 149 15))
                       (l (p 149 15) (p 183 15))
                       (ca (p 183 15) (p 185 13) (p 185 15))
                       (l (p 185 13) (p 185 0))
                       (p 185 0)
                       (p 185 0))                          ) )

;
; For workpieces that are not well-formed an unknown failure is produced:
;
; rfi> (wp (l (p 0 0) (p 0 12.5)) (l (p 0 13.5) (p 17 0)))   ; lines don't meet
; unknown
;
; For well-formed workpieces, like those in (test), normal forms are returned:
;
; rfi> (wp (p 0 0)                    ; calling the wp that needs normalization
;          (l (p 0 0) (p 0 12.5))
;          (l (p 0 12.5) (p 0 12.5))
;          (l (p 0 12.5) (p 27 44.5))
;          (p 27 44.5)
;          (l (p 27 44.5) (p 44 44.5))
;          (p 44 44.5)
;          (p 44 44.5)
;          (p 44 44.5)
;          (l (p 44 44.5) (p _a 20))
;          (l (p 44 20) (p 149 20))
;          (l (p 149 20) (p 149 15))
;          (l (p 149 15) (p 183 15))
;          (ca (p 183 15) (p 185 13) (p 185 15))
;          (l (p 185 13) (p 185 0))
;          (p 185 0)
;          (p 185 0) )
; (wp
;  (l (p 0 0) (p 0 12.5))
;  (l (p 0 12.5) (p 27 44.5))
;  (l (p 27 44.5) (p 44 44.5))
;  (l (p 44 44.5) (p 44 20))
;  (l (p 44 20) (p 149 20))
```

```
;  (l (p 149 20) (p 149 15))
;  (l (p 149 15) (p 183 15))
;  (ca (p 183 15) (p 185 13) (p 185 15))
;  (l (p 185 13) (p 185 0)) ) )
; (_a = 44)

; The normalized workpiece abbreviated with (n):
((n)                `(wp (l (p 0 0) (p 0 12.5))
                         (l (p 0 12.5) (p 27 44.5))
                         (l (p 27 44.5) (p 44 44.5))
                         (l (p 44 44.5) (p 44 20))
                         (l (p 44 20) (p 149 20))
                         (l (p 149 20) (p 149 15))
                         (l (p 149 15) (p 183 15))
                         (ca (p 183 15) (p 185 13) (p 185 15))
                         (l (p 185 13) (p 185 0))))

; For the (n) workpiece the specific operations return the following values:
;
; rfi> (wplength (n))
; 185
; rfi> (wptypes (n))
; (tup l l l l l l l ca l)
; rfi> (wpedges (n))
; (tup
;  (p 0 0)
;  (p 0 12.5)
;  (p 27 44.5)
;  (p 44 44.5)
;  (p 44 20)
;  (p 149 20)
;  (p 149 15)
;  (p 183 15)
;  (p 185 0) )
; rfi> (wpxes (n))
; (tup 0 12.5 44.5 44.5 20 20 15 15 0)
; rfi> (wpradius (n))
; 44.5
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Grouping and Verifying the Period System of the Elements: ;
;                  A Relfun Knowledge Base                  ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; (c) Michael Sintek  &  Werner Stein              March 1991

; A lot of basic physics/chemistry knowledge is encoded in the
; period system of the elements. The below Relfun knowledge base
; represents some properties of the elements and groups of elements.
; It was motivated by our work on enhanced WAM-like indexing
; techniques.
; Each entry has the following structure (the emphasis on metals
; derives from our application considerations):
;
; (per   <number>
;        (name <abbr> <name>)
;        (info <a metal or not a metal>
;              (eco <electron configuration of last three shells>)
;              <atom weight>
;              (oxi <list of possible oxidations>)))
;
; Using "domain-dependent" formulas, we compute groups (new notation)
; and verify the electron configuration of the entire knowledge base
; (see check-all-econs).


(hn (per 1
         (name H Hydrogen)
         (info n
               (eco 1 0 0)
               1.00794
               (oxi +1 -1))))

(hn (per 2
         (name He Helium)
         (info n
               (eco 2 0 0)
               4.0020602
               (oxi 0))))

(hn (per 3
         (name Li Lithium)
         (info n
               (eco 2 1 0)
               6.941
               (oxi +1))))

(hn (per 4
         (name Be Beryllium)
         (info n
               (eco 2 2 0)
               9.012182
               (oxi +2))))

(hn (per 5
         (name B Boron)
         (info n
               (eco 2 3 0)
               10.811
               (oxi +3))))
```

```
(hn (per 6
        (name C Carbon)
        (info n
              (eco 2 4 0)
              12.011
              (oxi +2 +4 -4))))

(hn (per 7
        (name N Nitrogen)
        (info n
              (eco 2 5 0)
              14.00674
              (oxi +1 +2 +3 +4 +5 -1 -2 -3))))

(hn (per 8
        (name O Oxygen)
        (info n
              (eco 2 6 0)
              15.9994
              (oxi -2))))

(hn (per 9
        (name F Fluorine)
        (info n
              (eco 2 7 0)
              18.9984032
              (oxi -1))))

(hn (per 10
        (name Ne Neon)
        (info n
              (eco 2 8 0)
              20.1797
              (oxi 0))))

(hn (per 11
        (name Na Sodium)
        (info m
              (eco 2 8 1)
              22.989768
              (oxi 1))))

(hn (per 12
        (name Mg Magnesium)
        (info m
              (eco 2 8 2)
              24.305
              (oxi +2))))

(hn (per 13
        (name Al Aluminium)
        (info m
              (eco 2 8 3)
              26.981539
              (oxi +3))))

(hn (per 14
        (name Si Silicon)
        (info n
              (eco 2 8 4)
              28.0855
              (oxi +2 +4 -4))))
```

```
(hn (per 15
        (name P Phosphorus)
        (info n
              (eco 2 8 5)
              30.97362
              (oxi +3 +5 -3))))

(hn (per 16
        (name S Sulfur)
        (info n
              (eco 2 8 6)
              32.066
              (oxi +4 +6 -2))))

(hn (per 17
        (name Cl Chlorine)
        (info n
              (eco 2 8 7)
              35.4527
              (oxi +1 +5 +7 -1))))
(hn (per 18
        (name Ar Argon)
        (info n
              (eco 2 8 8)
              39.948
              (oxi 0))))
(hn (per 19
        (name K Potassium)
        (info m
              (eco 8 8 1)
              39.0983
              (oxi +1))))

(hn (per 20
        (name Ca Calcium)
        (info m
              (eco 8 8 2)
              40.078
              (oxi +2))))


; ...


(hn (per 107
        (name Uns Unnilseptium)
        (info ?
              (eco 32 13 2)
              262
              (oxi))))
```

```
; some simple access functions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(hn (element _no)                    ; "generate" elements
  (per _no _id1 _id2))


(ft (metal)                          ; return metal names
  (per _x
       '(name _y _z)
       '(info m
              _a
              _b
              _c))
  _z)



; computing groups
;;;;;;;;;;;;;;;;;;

(ft (group 1) 1)                     ; find group no. of an element
(ft (group 2) 18)
(ft (group 3) 1)
(ft (group 4) 2)
(ft (group _no)
  (element _no)
  (<= 5 _no)
  (<= _no 10)
  (+ _no 8))
(ft (group 11) 1)
(ft (group 12) 2)
(ft (group _no)
  (element _no)
  (<= 13 _no)
  (<= _no 57)
  (+ 1 (mod (- _no 1) 18)))
(ft (group _no)                      ;Lanthanides
  (element _no)
  (<= 58 _no)
  (<= _no 71)
  (- _no 55))
(ft (group _no)
  (element _no)
  (<= 72 _no)
  (<= _no 89)
  (+ 1 (mod (- _no 69) 18)))
(ft (group _no)                      ;Actinides
  (element _no)
  (<= 90 _no)
  (<= _no 103)
  (- _no 87)))
(ft (group _no)
  (element _no)
  (<= 104 _no)
  (<= _no 107)
  (- _no 100)))
```

```
; verify electron configuration
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(hn (check-econ _no)              ; element no. must be equal to the
  (is _no (sum (econ _no))))      ; sum of (econ _no)



(ft (check-all-econs)             ; check knowledge base for inconsistencies
  (element _no)                   ; in electron configuration
  (<> 0 (check-econ1 _no))        ; enumerate the numbers of inconsistent
  _no)                            ; 'per' entries

(ft (check-all-econs) ok)         ; return ok if no more inconsistency is
                                  ; detected

(ft (check-econ1 _no) (- _no (sum (econ _no))))


(ft (offs _no)                    ; no. of inner shells before last three
  (<= 1 _no)                      ; (see per/3)
  (<= _no 18)
  0)
(ft (offs _no)
  (<= 19 _no)
  (<= _no 36)
  1)
(ft (offs _no)
  (<= 37 _no)
  (<= _no 54)
  2)
(ft (offs _no)
  (<= 55 _no)
  (<= _no 86)
  3)
(ft (offs _no)
  (<= 87 _no)
  (<= _no 107)
  4)


(ft (maxel 1) 2)                  ; max. no. of electrons on a shell
(ft (maxel 2) 8)
(ft (maxel 3) 18)
(ft (maxel 4) 32)


(ft (econ _no)                    ; compute electron configuration of an
  (per _no                        ; element (returns a list)
       _n
       '(info _m
              (eco _l1 _l2 _l3)
              _w
              _o))
  (lpref '(tup _l1 _l2 _l3) (offs _no)))


(ft (lpref _l 0) _l)              ; generate prefix of inner shells
(ft (lpref _l _of)                ; before last three
  (< 0 _of)
  (lpref (tup (maxel _of) | _l)  (- _of 1)))
```

```
; auxiliary functions
;;;;;;;;;;;;;;;;;;;;

(ft (tup) '(tup))                    ; evaluates its arguments!
(ft (tup _x | _ r) '(tup _x | _r))


(ft (sum (tup)) 0)                   ; return sum of a list
(ft (sum (tup _h | _t)) (+ _h (sum _t)))

(hn (<> _x _y) (> _x _y))
(hn (<> _x _y) (> _y _x))


(ft (mod _x _y) (< _x _y) _x)
(ft (mod _x _y) (>= _x _y) (mod (- _x _y) _y))



; sample dialog
;;;;;;;;;;;;;;;


; rfi> (group 10)
; 18
; rfi> (econ 10)
; (tup 2 8 0)
; rfi> (econ 107)
; (tup 2 8 18 32 32 13 2)
; rfi> (check-all-econs)
; ok
; rfi>
```

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
Stuhlsatzenhausweg 3
6600 Saarbrücken 11
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen oder die aktuelle Liste von erhältlichen Publikationen können bezogen werden von der oben angegebenen Adresse.

DFKI Publications

The following DFKI publications or the list of currently available publications can be ordered from the above address.

DFKI Research Reports

RR-90-01
Franz Baader: Terminological Cycles in KL-ONE-based Knowledge Representation Languages
33 pages

RR-90-02
Hans-Jürgen Bürckert: A Resolution Principle for Clauses with Constraints
25 pages

RR-90-03
Andreas Dengel, Nelson M. Mattos: Integration of Document Representation, Processing and Management
18 pages

RR-90-04
Bernhard Hollunder, Werner Nutt: Subsumption Algorithms for Concept Languages
34 pages

RR-90-05
Franz Baader: A Formal Definition for the Expressive Power of Knowledge Representation Languages
22 pages

RR-90-06
Bernhard Hollunder: Hybrid Inferences in KL-ONE-based Knowledge Representation Systems
21 pages

RR-90-07
Elisabeth André, Thomas Rist: Wissensbasierte Informationspräsentation:
Zwei Beiträge zum Fachgespräch Graphik und KI:

1. Ein planbasierter Ansatz zur Synthese illustrierter Dokumente
2. Wissensbasierte Perspektivenwahl für die automatische Erzeugung von 3D-Objektdarstellungen

24 pages

RR-90-08
Andreas Dengel: A Step Towards Understanding Paper Documents
25 pages

RR-90-09
Susanne Biundo: Plan Generation Using a Method of Deductive Program Synthesis
17 pages

RR-90-10
Franz Baader, Hans-Jürgen Bürckert, Bernhard Hollunder, Werner Nutt, Jörg H. Siekmann: Concept Logics
26 pages

RR-90-11
Elisabeth André, Thomas Rist: Towards a Plan-Based Synthesis of Illustrated Documents
14 pages

RR-90-12
Harold Boley: Declarative Operations on Nets
43 pages

RR-90-13
Franz Baader: Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles
40 pages

RR-90-14
Franz Schmalhofer, Otto Kühn, Gabriele Schmidt: Integrated Knowledge Acquisition from Text, Previously Solved Cases, and Expert Memories
20 pages

RR-90-15
Harald Trost: The Application of Two-level Morphology to Non-concatenative German Morphology
13 pages

RR-90-16
Franz Baader, Werner Nutt: Adding Homomorphisms to Commutative/Monoidal Theories, or: How Algebra Can Help in Equational Unification
25 pages

RR-91-01
Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, and Gert Smolka : On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations
20 pages

RR-91-02
Francesco Donini, Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, Werner Nutt: The Complexity of Existential Quantification in Concept Languages
22 pages

RR-91-03
B.Hollunder, Franz Baader: Qualifying Number Restrictions in Concept Languages
34 pages

RR-91-04
Harald Trost
X2MORF: A Morphological Component Based on Augmented Two-Level Morphology
19 pages

RR-91-05
Wolfgang Wahlster, Elisabeth André, Winfried Graf, Thomas Rist: Designing Illustrated Texts: How Language Production is Influenced by Graphics Generation.
17 pages

RR-91-06
Elisabeth André, Thomas Rist: Synthesizing Illustrated Documents
A Plan-Based Approach
11 pages

RR-91-07
Günter Neumann, Wolfgang Finkler: A Head-Driven Approach to Incremental and Parallel Generation of Syntactic Structures
13 pages

RR-91-08
Wolfgang Wahlster, Elisabeth André, Som Bandyopadhyay, Winfried Graf, Thomas Rist
WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation
23 pages

RR-91-09
Hans-Jürgen Bürckert, Jürgen Müller, Achim Schupeta
RATMAN and its Relation to Other Multi-Agent Testbeds
31 pages

RR-91-11
Bernhard Nebel
Belief Revision and Default Reasoning: Syntax-Based Approaches
37 pages

DFKI Technical Memos

TM-89-01
Susan Holbach-Weber: Connectionist Models and Figurative Speech
27 pages

TM-90-01
Som Bandyopadhyay: Towards an Understanding of Coherence in Multimodal Discourse
18 pages

TM-90-02
Jay C. Weber: The Myth of Domain-Independent Persistence
18 pages

TM-90-03
Franz Baader, Bernhard Hollunder: KRIS: Knowledge Representation and Inference System -System Description-
15 pages

TM-90-04
Franz Baader, Hans-Jürgen Bürckert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernhard Nebel, Werner Nutt, Hans-Jürgen Profitlich: Terminological Knowledge Representation: A Proposal for a Terminological Logic
7 pages

TM-91-01
Jana Köhler
Approaches to the Reuse of Plan Schemata in Planning Formalisms
52 pages

TM-91-02
Knut Hinkelmann
Bidirectional Reasoning of Horn Clause Programs: Transformation and Compilation
20 pages

TM-91-03
Otto Kühn, Marc Linster, Gabriele Schmidt
Clamping, COKAM, KADS, and OMOS: The Construction and Operationalization of a KADS Conceptual Model
20 pages

TM-91-04
Harold Boley
A sampler of Relational/Functional Definitions
12 pages

TM-91-05
Jay C. Weber, Andreas Dengel and Rainer Bleisinger
Theoretical Consideration of Goal Recognition Aspects for Understanding Information in Business Letters
10 pages

DFKI Documents

D-89-01
Michael H. Malburg, Rainer Bleisinger:
HYPERBIS: ein betriebliches Hypermedia-Informationssystem
43 Seiten

D-90-01
DFKI Wissenschaftlich-Technischer Jahresbericht 1989
45 pages

D-90-02
Georg Seul: Logisches Programmieren mit Feature -Typen
107 Seiten

D-90-03
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Abschlußbericht des Arbeitspaketes PROD
36 Seiten

D-90-04
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: STEP: Überblick über eine zukünftige Schnittstelle zum Produktdatenaustausch
69 Seiten

D-90-05
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Formalismus zur Repräsentation von Geo-metrie- und Technologieinformationen als Teil eines Wissensbasierten Produktmodells
66 Seiten

D-90-06
Andreas Becker: The Window Tool Kit
66 Seiten

D-91-01
Werner Stein , Michael Sintek
Relfun/X - An Experimental Prolog Implementation of Relfun
48 pages

D-91-03
Harold Boley, Klaus Elsbernd, Hans-Günther Hein, Thomas Krause
RFM Manual: Compiling RELFUN into the Relational/Functional Machine
43 pages

A Sampler of Relational/Functional Definitions

Harold Boley (Ed.)

TM-91-04
Technical Memo