Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-91-15

µCAD2NC:

A Declarative Lathe-Workplanning Model Transforming CAD-like Geometries into Abstract NC Programs

Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch, Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, Frank Steinle

November 1991

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11
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. The shareholder companies are Daimler Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Krupp-Atlas, Mannesmann-Kienzle, Philips, Sema Group Systems, Siemens and Siemens-Nixdorf. 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

µCAD2NC:
A Declarative Lathe-Workplanning Model Transforming CAD-like Geometries into Abstract NC Programs

Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch, Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, Frank Steinle

DFKI-D-91-15

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-8902 C4).

μCAD2NC: A Declarative Lathe-Workplanning Model Transforming CAD-like Geometries into Abstract NC Programs*

Harold Boley Philipp Hanschke
Martin Harm Knut Hinkelmann Thomas Labisch
Manfred Meyer Jörg Müller Thomas Oltzen
Michael Sintek Werner Stein Frank Steinle

DFKI Kaiserslautern
W-6750 Kaiserslautern, F.R. Germany

November 1991

Abstract

μCAD2NC is a knowledge-based system generating workplans for idealized lathe CNC machines. It transforms CAD-like geometries of rotational-symmetric workpieces into abstract NC programs, using declarative term representations for all processing steps. The system has been developed using COLAB, a hybrid-knowledge compilation laboratory which integrates the power of forward and backward reasoning (incl. functional programming), constraint propagation, and taxonomic classification. The focus of this work is on exemplifying techniques of the hybrid, declarative COLAB formalisms for the central subtasks of CAD-to-NC transformations.

Contents

1 Introduction

The μCAD2NC (micro-CAD-to-NC) system solves the following simplified model version of a real-world problem:

> Given the geometry of a rotational-symmetric workpiece, generate abstract NC macros for rough-turning the workpiece on an abstract CNC lathe machine (cf. figures in Appendix E).

This model performs the most interesting **central** phases of CAD-to-NC transformation; writing a front-end for converting real CAD data to our knowledge-base representation, and a backend for converting our NC macros to programs for a real CNC machine would be a routine task. The intention behind μCAD2NC is not to provide a polished solution tuned for production: instead, it enables us to study how AI techniques and formalisms can be combined to a non-toy prototype. μCAD2NC is implemented in COLAB [Boley *et al.*, 1991], a knowledge-processing system integrating forward reasoning, backward reasoning (including functions), constraints, and terminologies. In μCAD2NC the subformalisms of COLAB collaborate in a prototypical synergetic manner.

Regarding efficiency, we concentrate on principal algorithmic issues (including evaluation functions for heuristics) of applying COLAB as a hybrid-knowledge compilation laboratory to NC-planning, exploiting knowledge from the mechanical-engineering domain of lathe turning. Algorithmic efficiency has been gained via functional-programming methods such as accumulator parameters, 'horizontal' compilation techniques (source-to-source transformations) such as preclassification of concept definitions, and our 'vertical' WAM compiler. Employing a fast COMMON LISP (e.g., the Symbolics UX1200S board) also results in good low-level efficiency of our declarative, very-high-level formulations.

2 Short Introduction to Our Approach

The input to a production planning system is a very 'elementary' description of a workpiece as it comes from a CAD system. Geometrical descriptions of the workpiece's surfaces and topological neighborhood relations are the central parts of this representation. If possible at all, production planning with these data starting from (nearly) first principles would require very complex algorithms. Thus, planning strategies on such a detailed level are neither available nor do they make sense. Instead, human planners have a library of *skeletal plans* in their minds [Schmalhofer *et al.*, 1991]. Each of these plans is accessed via a more or less abstract description of a characteristic (part of a) workpiece, which is called a *workpiece feature* [Klauck *et al.*, 1991] . A feature thus associates a workpiece model (geometry/topology, technology) with the corresponding manufacturing method (NC program, chucking, toolchange).

Therefore, the first step of μCAD2NC is the generation of an abstract feature description from the elementary workpiece data; the features obtained characterize the workpiece with respect to its production (Section 3). In the second step, the skeletal plans (associated with the features) are retrieved and merged, resulting in an abstract NC program.

In μCAD2NC we use a transformational approach whose processing steps all map between explicit intermediate term representations (with hybrid "plug-in" components), which contributes to the flexibility of the system. The whole NC-planning process of μCAD2NC is illustrated in Appendix E: Starting from the initial workpiece representation the feature descriptions are aggregated, which themselves form the basis for skeletal-plan retrieval. Symbolic skeletal-plan execution is simulated to obtain a workplan consisting of the desired NC macros.

3 Performing the Workpiece Classification in COLAB

The various subtasks of workpiece classification require a number of specialized reasoning mechanisms, which are integrated in the compilation laboratory COLAB. Feature aggregation in μCAD2NC is performed by the forward reasoning system of COLAB, FORWARD [Hinkelmann, 1991], together with the system for representing taxonomic knowledge, TAXON [Baader and Hanschke, 1991a]. The derived features are then collected by a program written in RELFUN [Boley, 1990], the backward reasoning and functional component of COLAB.

FORWARD [Hinkelmann, 1991] is a declarative rule-based system with Horn clauses as its basic representation scheme, which is tightly coupled with RELFUN to achieve bidirectional reasoning. It offers two independent evaluation procedures: The first interprets bottom-up rules directly using a magic set transformation for goal-directed reasoning. The second transforms bottom-up and bidirectional rules to RELFUN Horn clauses which are finally compiled into an extended RFM-System with a special forward code area. The latter version is currently used in the μCAD2NC system.

TAXON [Baader and Hanschke, 1991a] is a KL-ONE-like knowledge representation system. It provides two subformalisms: one to define and reason about terminologies, called Tbox, and another (called Abox) to reason about assertional knowledge. A terminology consists of a set of intensional concept definitions, which are arranged in a *subsumption hierarchy* (actually a directed acyclic graph) by the *classification service*. In the Abox individuals instantiate concepts and are related to each other by attributes and roles. This assertional knowledge is used to determine the most specific concepts in the subsumption hierarchy to which the individuals belong (*realization service*).

Following the distinction between concepts and instances it is rather natural to define all the possible features and surfaces as concepts in TAXON's Tbox and to represent a single case, i.e. a workpiece, by assertions in the Abox. At a first glance, determining the most specific problem class related to a workpiece (i.e. the first step of μCAD2NC) could be mapped to the realization service of TAXON. But TAXON alone is not suitable for the feature recognition task for two reasons:

- Most features cover a number of surfaces. This means that it is natural to define a feature as consisting of simpler features (and having some additional requirements, e.g. neighbourhood). Thus, finding such a feature means to find individuals representing the components and to generate a new individual aggregating the simpler features using e.g. part-of attributes. But the automatic generation of new instances is not a standard operation in terminological systems. For a rule system, however, such an aggregation of instances for building new objects is not a problem.
- The second reason is that terminological systems aim at decision procedures for their reasoning services, which restricts their expressiveness. For example, it is not possible to deal with concrete domains (e.g. real numbers) and varying size aspects (e.g. sequences) in *one* concept language in a reasonable way, without having an undecidable subsumption problem [Baader and Hanschke, 1991b]. The way out of this problem is to exclude one of these aspects from the terminological formalism and deal with the other in a (tightly coupled) rule language.

The cooperation of FORWARD and TAXON combines the general-purpose reasoning power of rule-based systems with the inheritance abstraction provided by terminological systems together with their ability to check (concept) definitions for plausibility.

How does this interaction work?

The input is a term representation of CAD geometries (in the logic-programming tradition of declarative pictures, graphics, and geometries [Kowalski, 1982; Helm and Marriott, 1986; Pereira, 1986]), where each elementary surface region becomes an attribute term asserted individually in COLAB's Abox. For example in the sample workpiece representation in Chapter 5.1 the truncated cone `tr38` and the neighbouring cylinder **cyl39** are represented as two attribute terms:

```
(attrterm (truncone tr38 (tup (tup center1 50)
                              (tup center2 60)
                              (tup radius1 25)
                              (tup radius2 40))))
(attrterm (cylinder cyl39 (tup (tup center1 60)
                               (tup center2 70)
                               (tup radius1 40)
                               (tup radius2 40))))
```

Here **attrterm** is a tag indicating attribute terms. **truncone** and **cylinder** are concepts defined in the terminology. Both surfaces are represented by four parameters to enable subsumption; thus one radius of the cylinders is redundant and could be dropped in principal. The tag **tup** can be regarded as list constructor.

To generate the feature abstraction the rule system starts bottom up with such a collection of attribute terms by asserting features into TAXON's Abox. Since the features are already defined in TAXON it would be superfluous to repeat the whole number of definitions as rules in FORWARD. Therefore the rules are very general mentioning in the ideal case only the most general features (in the subsumption hierarchy) ranging over corresponding numbers of surfaces. As soon as a new feature instance or additional information about an already existing instance is asserted, TAXON computes its most special concept associations using the realization service. This information gain, resulting in new facts in the Abox, can again trigger rules to derive further features building on the feature just found.

In our μCAD2NC model a groove is simply defined as an aggregation of two shoulders with common ground. If, for example, this aggregation is performed by the rule system, TAXON may assert (using the realization service) that the groove is actually an insertion. This new fact triggers every rule with premises referring to insertions, grooves, and other more general features. This is implemented in the current system in a very naive way. Before each forward reasoning step the FORWARD system asks for the concept closure of an instance, i.e. the whole set of concepts the instance belongs to. Rule activation then proceeds with this whole set of general concept associations.

To assert facts into TAXON's Abox an operator **add-data** is introduced which has three arguments: the name of an instance, the concept the instance belongs to, and a list of attribute-value pairs:

```
(add-data <concept-name>
          <instance-name>
          (tup (tup <attr1> <val1>)
               ...
               (tup <attrN> <valN>))
```

To retrieve information from TAXON to satisfy a rule's premises a corresponding operator **data** is defined.

The remainder of this section sketches how the feature abstractions of the workpiece represented in the Abox are converted in order to serve the skeletal plan retrieval and merging. The resulting representation is less redundant, deals only with one of the feature abstractions, and is term oriented.

The transformation consists of three main steps:

1. Partially sorting the list of features: A feature ranges over a number of surfaces. A feature f *covers* a feature g iff the set of surfaces belonging to f is a superset of g.

2. Recognition of the most complex features: The mentioned partial ordering on features is the basis for removing redundant information in the feature abstractions. For example a groove consists of a left and a right shoulder. The information about the shoulders would be dropped in this step, while the components (two flanks and a ground) would be kept.

3. Constructing a classified workpiece from the remaining features: A representation of a classified workpiece contains the highest radius of the workpiece and a list of nested features occurring in this workpiece. The nested features are described by their names and a list of (possibly again nested) features. This list of nested features is commutative, reflecting the partial ordering of point 1.

Classified workpiece:

```
(cwp
   <radius>
   (tup (nft ...) ... (nft ...)))
```

Nested feature:

```
(nft
   <feature>
   (tup (nft ...) ... (nft ...)))
```

All these refining steps are implemented via the RELFUN [Boley, 1990] system, a relational-functional language integrated into COLAB.

According to our modular-transformational approach, the first step could alternatively there are a lot of restrictions, which holder to use for which plate, which kind of plate geometry to use for which workpiece contour etc. As an example, the following COLAB code represents the definition of a primitive constraint named **process-edge-angle** between the process (finishing or roughing) and symbolic classes of edge-angles:

```
(pc process-edge-angle (pr ea) (processes edge-angles)
  (roughing small-edge-angles)
  (roughing medium-edge-angles)
  (finishing medium-edge-angles)
  (finishing big-edge-angles))
```

All these restrictions, which need not to be only binary ones, constrain the search space for valid combinations of values for the different problem variables. Therefore, it seems to be a natural way to use a constraint propagation system to perform this subtask. The constraint system CONTAX provides an efficient mechanism to

1. formulate the constraint problem by defining the problem variables, their domains, and the constraints (relations) ranging over them; and to
2. propagate given (initial) value restrictions through this network of constraints.

The fact, that domains may be defined hierarchically instead of explicitly enumerating all the elements of the domain, is very useful for the use of CONTAX within μCAD2NC, since the domains of lathe tools and holders can be hierarchically structured in a very natural way. For example, the domain of lathe

5 Sample Session

In order to illustrate the transformation processes, one consultation of a μCAD2NC session is presented in this chapter.

5.1 Demo Workpiece

The workpiece description of the μCAD2NC sample session consists of geometrical and topological information about the workpiece's surfaces. Because each rotational-symmetric surface is a specialization of a truncated cone, it can be described by four attributes: two co-ordinates and two radii.

```
(attrterm (circle circ33 (tup (tup center1 0)
                              (tup center2 0)
                              (tup radius1 0)
                              (tup radius2 25))))
(attrterm (cylinder cyl34 (tup (tup center1 0)
                               (tup center2 40)
                               (tup radius1 25)
                               (tup radius2 25))))
(attrterm (ring rng35 (tup (tup center1 40)
                           (tup center2 40)
                           (tup radius1 25)
                           (tup radius2 20))))
(attrterm (cylinder cyl36 (tup (tup center1 40)
                               (tup center2 50)
                               (tup radius1 20)
                               (tup radius2 20))))
(attrterm (ring rng37 (tup (tup center1 50)
                           (tup center2 50)
                           (tup radius1 20)
                           (tup radius2 25))))
(attrterm (truncone tr38 (tup (tup center1 50)
                              (tup center2 60)
                              (tup radius1 25)
                              (tup radius2 40))))
(attrterm (cylinder cyl39 (tup (tup center1 60)
                               (tup center2 70)
                               (tup radius1 40)
                               (tup radius2 40))))
(attrterm (ring rng40 (tup (tup center1 70)
                           (tup center2 70)
                           (tup radius1 40)
                           (tup radius2 30))))
(attrterm (cylinder cyl41 (tup (tup center1 70)
                               (tup center2 110)
                               (tup radius1 30)
                               (tup radius2 30))))
```

```
(attrterm (ring rng42 (tup (tup center1 110)
                           (tup center2 110)
                           (tup radius1 30)
                           (tup radius2 20))))
(attrterm (cylinder cyl43 (tup (tup center1 110)
                               (tup center2 120)
                               (tup radius1 20)
                               (tup radius2 20))))
(attrterm (ring rng44 (tup (tup center1 120)
                           (tup center2 120)
                           (tup radius1 20)
                           (tup radius2 25))))
(attrterm (cylinder cyl45 (tup (tup center1 120)
                               (tup center2 150)
                               (tup radius1 25)
                               (tup radius2 25))))
(attrterm (circle circ46 (tup (tup center1 150)
                              (tup center2 150)
                              (tup radius1 25)
                              (tup radius2 0))))
(fact (neighbor circ33 cyl34))
(fact (neighbor cyl34 rng35))
(fact (neighbor rng35 cyl36))
(fact (neighbor cyl36 rng37))
(fact (neighbor rng37 tr38))
(fact (neighbor tr38 cyl39))
(fact (neighbor cyl39 rng40))
(fact (neighbor rng40 cyl41))
(fact (neighbor cyl41 rng42))
(fact (neighbor rng42 cyl43))
(fact (neighbor cyl43 rng44))
(fact (neighbor rng44 cyl45))
(fact (neighbor cyl45 circ46))
```

5.2 Detailed Trace

Here we give a complete trace of the μCAD2NC sample session showing the "operational sematics" of our CAD-to-NC transformation model.[1] Except from the LaTeXpage formatting, the trace is reproduced verbatim from an actual COLAB demo run.

```
colab,fw> (demo/f)

derived-fact-asserted-into-taxon-abox
(add-data
 shoulder
 shoulder-rng35-lts-cyl36-cyl36
 (tup
  (tup ground lts-cyl36-cyl36)
  (tup flank rng35)
  (tup leftmost rng35)
  (tup rightmost cyl36) ) )

derived-fact-asserted-into-taxon-abox
(add-data
 longturningsurface
 lts-cyl36-cyl36
 (tup (tup radius 20) (tup leftmost cyl36) (tup rightmost cyl36)) )

derived-fact-asserted-into-taxon-abox
(add-data
 shoulder
 shoulder-lts-cyl36-cyl36-rng37
 (tup
  (tup ground lts-cyl36-cyl36)
  (tup flank rng37)
  (tup leftmost cyl36)
  (tup rightmost rng37) ) )

derived-fact-asserted-into-taxon-abox
(add-data
 shoulder
 shoulder-rng40-lts-cyl41-circ46
 (tup
  (tup ground lts-cyl41-circ46)
  (tup flank rng40)
  (tup leftmost rng40)
  (tup rightmost circ46) ) )
```

[1] Of course, also selective traces are possible and everything except the final ANC program would be hidden to endusers.

```
derived-fact-asserted-into-taxon-abox
(add-data
 longturningsurface
 lts-cyl41-circ46
 (tup (tup radius 30) (tup leftmost cyl41) (tup rightmost circ46)) )

derived-fact-asserted-into-taxon-abox
(add-data
 shoulder
 shoulder-rng42-lts-cyl43-cyl43
 (tup
  (tup ground lts-cyl43-cyl43)
  (tup flank rng42)
  (tup leftmost rng42)
  (tup rightmost cyl43) ) )

derived-fact-asserted-into-taxon-abox
(add-data
 shoulder
 shoulder-rng42-lts-cyl43-circ46
 (tup
  (tup ground lts-cyl43-circ46)
  (tup flank rng42)
  (tup leftmost rng42)
  (tup rightmost circ46) ) )

derived-fact-asserted-into-taxon-abox
(add-data
 longturningsurface
 lts-cyl43-circ46
 (tup (tup radius 25) (tup leftmost cyl43) (tup rightmost circ46)) )

derived-fact-asserted-into-taxon-abox
(add-data
 shoulder
 shoulder-lts-cyl43-cyl43-rng44
 (tup
  (tup ground lts-cyl43-cyl43)
  (tup flank rng44)
  (tup leftmost cyl43)
  (tup rightmost rng44) ) )
```

```
derived-fact-asserted-into-taxon-abox
(add-data
 longturningsurface
 lts-cyl43-cyl43
 (tup (tup radius 20) (tup leftmost cyl43) (tup rightmost cyl43)) )

derived-fact-asserted-into-taxon-abox
(add-data
 shoulder
 shoulder-lts-circ33-rng37-tr38
 (tup
  (tup ground lts-circ33-rng37)
  (tup flank tr38)
  (tup leftmost circ33)
  (tup rightmost tr38) ) )

derived-fact-asserted-into-taxon-abox
(add-data
 longturningsurface
 lts-circ33-rng37
 (tup (tup radius 25) (tup leftmost circ33) (tup rightmost rng37)) )

derived-fact-asserted-into-taxon-abox
(add-data
 groove
 groove-rng35-lts-cyl36-cyl36-rng37
 (tup
  (tup leftflank rng35)
  (tup ground lts-cyl36-cyl36)
  (tup rightflank rng37)
  (tup leftmost rng35)
  (tup rightmost rng37) ) )

derived-fact-asserted-into-taxon-abox
(add-data
 groove
 groove-rng42-lts-cyl43-cyl43-rng44
 (tup
  (tup leftflank rng42)
  (tup ground lts-cyl43-cyl43)
  (tup rightflank rng44)
  (tup leftmost rng42)
  (tup rightmost rng44) ) )
```

```
features:

(tup
 (desc-tc
  rng35
  (tup (tup center1 40) (tup center2 40) (tup radius1 25) (tup radius2 20)) )
 (asc-tc
  rng37
  (tup (tup center1 50) (tup center2 50) (tup radius1 20) (tup radius2 25)) )
 (desc-tc
  rng40
  (tup (tup center1 70) (tup center2 70) (tup radius1 40) (tup radius2 30)) )
 (desc-tc
  rng42
  (tup (tup center1 110) (tup center2 110) (tup radius1 30) (tup radius2 20)) )
 (asc-tc
  rng44
  (tup (tup center1 120) (tup center2 120) (tup radius1 20) (tup radius2 25)) )
 (desc-tc
  [illegible]
  (tup (tup center1 150) (tup center2 150) (tup radius1 25) (tup radius2 0)) )
 (asc-tc
  circ33
  (tup (tup center1 0) (tup center2 0) (tup radius1 0) (tup radius2 25)) )
 (asc-tc
  tr38
  (tup (tup center1 50) (tup center2 60) (tup radius1 25) (tup radius2 40)) )
 (data
  shoulder
  shoulder-rng35-lts-cyl36-cyl36
  (tup
   (tup ground lts-cyl36-cyl36)
   (tup flank rng35)
   (tup leftmost rng35)
   (tup rightmost cyl36) ) )
 (data
  longturningsurface
  lts-cyl36-cyl36
  (tup (tup radius 20) (tup leftmost cyl36) (tup rightmost cyl36)) )
 (data
  shoulder
  shoulder-lts-cyl36-cyl36-rng37
  (tup
   (tup ground lts-cyl36-cyl36)
   (tup flank rng37)
   (tup leftmost cyl36)
   (tup rightmost rng37) ) )
```

```
(data
 shoulder
 shoulder-rng40-lts-cyl41-circ46
 (tup
  (tup ground lts-cyl41-circ46)
  (tup flank rng40)
  (tup leftmost rng40)
  (tup rightmost circ46) ) )
(data
 longturningsurface
 lts-cyl41-circ46
 (tup (tup radius 30) (tup leftmost cyl41) (tup rightmost circ46)) )
(data
 shoulder
 shoulder-rng42-lts-cyl43-cyl43
 (tup
  (tup ground lts-cyl43-cyl43)
  (tup flank rng42)
  (tup leftmost rng42)
  (tup rightmost cyl43) ) )
(data
 shoulder
 shoulder-rng42-lts-cyl43-circ46
 (tup
  (tup ground lts-cyl43-circ46)
  (tup flank rng42)
  (tup leftmost rng42)
  (tup rightmost circ46) ) )
(data
 longturningsurface
 lts-cyl43-circ46
 (tup (tup radius 25) (tup leftmost cyl43) (tup rightmost circ46)) )
(data
 shoulder
 shoulder-lts-cyl43-cyl43-rng44
 (tup
  (tup ground lts-cyl43-cyl43)
  (tup flank rng44)
  (tup leftmost cyl43)
  (tup rightmost rng44) ) )
(data
 longturningsurface
 lts-cyl43-cyl43
 (tup (tup radius 20) (tup leftmost cyl43) (tup rightmost cyl43)) )
(data
 shoulder
 shoulder-lts-circ33-rng37-tr38
 (tup
  (tup ground lts-circ33-rng37)
  (tup flank tr38)
  (tup leftmost circ33)
  (tup rightmost tr38) ) )
```

```
(data
 longturningsurface
 lts-circ33-rng37
 (tup (tup radius 25) (tup leftmost circ33) (tup rightmost rng37)) )
(data
 groove
 groove-rng35-lts-cyl36-cyl36-rng37
 (tup
  (tup leftflank rng35)
  (tup ground lts-cyl36-cyl36)
  (tup rightflank rng37)
  (tup leftmost rng35)
  (tup rightmost rng37) ) )
(data
 groove
 groove-rng42-lts-cyl43-cyl43-rng44
 (tup
  (tup leftflank rng42)
  (tup ground lts-cyl43-cyl43)
  (tup rightflank rng44)
  (tup leftmost rng42)
  (tup rightmost rng44) ) ) )
```

```
classified workpiece:

(cwp
 40
 (tup
  (nft
   (lsh (flk (tup (p 70 40) (p 70 30))) (grd (tup (p 70 30) (p 150 30))))
   (tup
    (nft
     (lsh (flk (tup (p 110 30) (p 110 25))) (grd (tup (p 110 25) (p 150 25))))
     (tup
      (nft
       (grv
        (flk (tup (p 110 25) (p 110 20)))
        (grd (tup (p 110 20) (p 120 20)))
        (flk (tup (p 120 20) (p 120 25))) )
       (tup) ) ) ) ) )
  (nft
   (rsh (grd (tup (p 0 25) (p 50 25))) (flk (tup (p 50 25) (p 60 40))))
   (tup
    (nft
     (grv
      (flk (tup (p 40 25) (p 40 20)))
      (grd (tup (p 40 20) (p 50 20)))
      (flk (tup (p 50 20) (p 50 25))) )
     (tup) ) ) ) ) )
```

```
contax tool selection:
  arguments: roughing high-alloy-steel normal 0 90 left
  propagating...
  results: (tup (tup dnmm-71 tmaxp-pdl93) (tup rcmx tmaxp-prl40)
                (tup rcmx tmaxp-prl30) (tup tnmm-71 tmaxp-ptl90))


contax tool selection:
  arguments: roughing high-alloy-steel normal 0 90 left
  propagating...
  results: (tup (tup dnmm-71 tmaxp-pdl93) (tup rcmx tmaxp-prl40)
                (tup rcmx tmaxp-prl30) (tup tnmm-71 tmaxp-ptl90))
    (grd (tup (p 40 20) (p 50 20)))
      (flk (tup (p 50 20) (p 50 25))) )


contax tool selection:
  arguments: roughing high-alloy-steel normal 90 90 right
  propagating...
  results: (tup (tup dnmm-71 tmaxp-pdr93) (tup rcmx tmaxp-prr30)
                (tup tnmm-71 tmaxp-ptr90))


contax tool selection:
  arguments: roughing high-alloy-steel normal 90 90 left
  propagating...
  results: (tup (tup dnmm-71 tmaxp-pdl93) (tup rcmx tmaxp-prl40)
                (tup rcmx tmaxp-prl30) (tup tnmm-71 tmaxp-ptl90))


contax tool selection:
  arguments: roughing high-alloy-steel normal 0 60 right
  propagating...
  results: (tup (tup dnmm-71 tmaxp-pdr93) (tup tnmm-71 tmaxp-ptr90)
                (tup tnmm-71 tmaxp-ptn60) (tup rcmx tmaxp-prr30))


contax tool selection:
  arguments: roughing high-alloy-steel normal 90 90 right
  propagating...
  results: (tup (tup dnmm-71 tmaxp-pdr93) (tup rcmx tmaxp-prr30)
                (tup tnmm-71 tmaxp-ptr90))


contax tool selection:
  arguments: roughing high-alloy-steel normal 90 90 left
  propagating...
  results: (tup (tup dnmm-71 tmaxp-pdl93) (tup rcmx tmaxp-prl40)
                (tup rcmx tmaxp-prl30) (tup tnmm-71 tmaxp-ptl90))
```

```
skeletal plan:

(skp
 40
 (com
  (tup
   (seq
    (tup
     (alt
      (tup
       (roughing
        (tool dnmm-71 tmaxp-pd193)
        left
        (geo (tup (p 70 40) (p 70 30) (p 150 30))) )
       (roughing
        (tool rcmx tmaxp-prl40)
        left
        (geo (tup (p 70 40) (p 70 30) (p 150 30))) )
       (roughing
        (tool rcmx tmaxp-prl30)
        left
        (geo (tup (p 70 40) (p 70 30) (p 150 30))) )
       (roughing
        (tool tnmm-71 tmaxp-ptl90)
        left
        (geo (tup (p 70 40) (p 70 30) (p 150 30))) ) ) )
     (alt
      (tup
       (roughing
        (tool dnmm-71 tmaxp-pd193)
        left
        (geo (tup (p 110 30) (p 110 25) (p 150 25))) )
       (roughing
        (tool rcmx tmaxp-prl40)
        left
        (geo (tup (p 110 30) (p 110 25) (p 150 25))) )
       (roughing
        (tool rcmx tmaxp-prl30)
        left
        (geo (tup (p 110 30) (p 110 25) (p 150 25))) )
       (roughing
        (tool tnmm-71 tmaxp-ptl90)
        left
        (geo (tup (p 110 30) (p 110 25) (p 150 25))) ) ) )
     (alt
      (tup
       (seq
        (tup
         (alt
          (tup
           (roughing
```

```
        (tool dnmm-71 tmaxp-pdl93)
        left
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
       (roughing
        (tool rcmx tmaxp-prl40)
        left
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
       (roughing
        (tool rcmx tmaxp-prl30)
        left
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
       (roughing
        (tool tnmm-71 tmaxp-ptl90)
        left
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) ) ) )
     (alt
      (tup
       (roughing
        (tool dnmm-71 tmaxp-pdr93)
        right
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
       (roughing
        (tool rcmx tmaxp-prr30)
        right
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
       (roughing
        (tool tnmm-71 tmaxp-ptr90)
        right
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) ) ) ) ) )
   (seq
    (tup
     (alt
      (tup
       (roughing
        (tool dnmm-71 tmaxp-pdr93)
        right
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
       (roughing
        (tool rcmx tmaxp-prr30)
        right
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
       (roughing
        (tool tnmm-71 tmaxp-ptr90)
        right
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) ) ) )
     (alt
      (tup
       (roughing
        (tool dnmm-71 tmaxp-pdl93)
        left
        (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
       (roughing
```

```
         (tool rcmx tmaxp-prl40)
         left
         (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
        (roughing
         (tool rcmx tmaxp-prl30)
         left
         (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
        (roughing
         (tool tnmm-71 tmaxp-ptl90)
         left
         (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) ) ) ) ) ) ) ) ) )
(seq
 (tup
  (alt
   (tup
    (roughing
     (tool dnmm-71 tmaxp-pdr93)
     right
     (geo (tup (p 0 25) (p 50 25) (p 60 40))) )
    (roughing
     (tool tnmm-71 tmaxp-ptr90)
     right
     (geo (tup (p 0 25) (p 50 25) (p 60 40))) )
    (roughing
     (tool tnmm-71 tmaxp-ptn60)
     right
     (geo (tup (p 0 25) (p 50 25) (p 60 40))) )
    (roughing
     (tool rcmx tmaxp-prr30)
     right
     (geo (tup (p 0 25) (p 50 25) (p 60 40))) ) ) )
  (alt
   (tup
    (seq
     (tup
      (alt
       (tup
        (roughing
         (tool dnmm-71 tmaxp-pdl93)
         left
         (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
        (roughing
         (tool rcmx tmaxp-prl40)
         left
         (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
        (roughing
         (tool rcmx tmaxp-prl30)
         left
         (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
        (roughing
         (tool tnmm-71 tmaxp-ptl90)
         left
```

```
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) )
   (alt
    (tup
     (roughing
      (tool dnmm-71 tmaxp-pdr93)
      right
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
     (roughing
      (tool rcmx tmaxp-prr30)
      right
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
     (roughing
      (tool tnmm-71 tmaxp-ptr90)
      right
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) ) ) )
 (seq
  (tup
   (alt
    (tup
     (roughing
      (tool dnmm-71 tmaxp-pdr93)
      right
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
     (roughing
      (tool rcmx tmaxp-prr30)
      right
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
     (roughing
      (tool tnmm-71 tmaxp-ptr90)
      right
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) )
   (alt
    (tup
     (roughing
      (tool dnmm-71 tmaxp-pdl93)
      left
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
     (roughing
      (tool rcmx tmaxp-prl40)
      left
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
     (roughing
      (tool rcmx tmaxp-prl30)
      left
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
     (roughing
      (tool tnmm-71 tmaxp-ptl90)
      left
      (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) ) ) ) ) ) ) ) ) ) )
```

```
qualitative simulation:

(tup
 (roughing
  (tool dnmm-71 tmaxp-pdl93)
  left
  (geo (tup (p 70 40) (p 70 30) (p 150 30))) )
 (roughing
  (tool dnmm-71 tmaxp-pdl93)
  left
  (geo (tup (p 110 30) (p 110 25) (p 150 25))) )
 (roughing
  (tool dnmm-71 tmaxp-pdl93)
  left
  (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
 tup )
4.4->
(skp
 40
 (com
  (tup
   (alt
    (tup
     (roughing
      (tool dnmm-71 tmaxp-pdr93)
      right
      (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
     (roughing
      (tool rcmx tmaxp-prr30)
      right
      (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
     (roughing
      (tool tnmm-71 tmaxp-ptr90)
      right
      (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) ) ) )
   (seq
    (tup
     (alt
      (tup
       (roughing
        (tool dnmm-71 tmaxp-pdr93)
        right
        (geo (tup (p 0 25) (p 50 25) (p 60 40))) )
       (roughing
        (tool tnmm-71 tmaxp-ptr90)
        right
        (geo (tup (p 0 25) (p 50 25) (p 60 40))) )
       (roughing
        (tool tnmm-71 tmaxp-ptn60)
        right
        (geo (tup (p 0 25) (p 50 25) (p 60 40))) )
```

```
  (roughing
   (tool rcmx tmaxp-prr30)
   right
   (geo (tup (p 0 25) (p 50 25) (p 60 40))) ) ) )
(alt
 (tup
  (seq
   (tup
    (alt
     (tup
      (roughing
       (tool dnmm-71 tmaxp-pdl93)
       left
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
      (roughing
       (tool rcmx tmaxp-prl40)
       left
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
      (roughing
       (tool rcmx tmaxp-prl30)
       left
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
      (roughing
       (tool tnmm-71 tmaxp-ptl90)
       left
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) )
    (alt
     (tup
      (roughing
       (tool dnmm-71 tmaxp-pdr93)
       right
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
      (roughing
       (tool rcmx tmaxp-prr30)
       right
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
      (roughing
       (tool tnmm-71 tmaxp-ptr90)
       right
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) ) ) )
  (seq
   (tup
    (alt
     (tup
      (roughing
       (tool dnmm-71 tmaxp-pdr93)
       right
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
      (roughing
       (tool rcmx tmaxp-prr30)
       right
       (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
```

```
          (roughing
           (tool tnmm-71 tmaxp-ptr90)
           right
           (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) )
         (alt
          (tup
           (roughing
            (tool dnmm-71 tmaxp-pdl93)
            left
            (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
           (roughing
            (tool rcmx tmaxp-prl40)
            left
            (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
           (roughing
            (tool rcmx tmaxp-prl30)
            left
            (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
           (roughing
            (tool tnmm-71 tmaxp-ptl90)
            left
            (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) ) ) ) ) ) ) ) ) )


(tup
 (roughing
  (tool dnmm-71 tmaxp-pdr93)
  right
  (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
 tup
 (roughing
  (tool dnmm-71 tmaxp-pdr93)
  right
  (geo (tup (p 0 25) (p 50 25) (p 60 40))) )
 (roughing
  (tool dnmm-71 tmaxp-pdr93)
  right
  (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
 tup )
3.7->
(skp
 40
 (alt
  (tup
   (roughing
    (tool dnmm-71 tmaxp-pdl93)
    left
    (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
   (roughing
    (tool rcmx tmaxp-prl40)
    left
    (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
```

```
   (roughing
    (tool rcmx tmaxp-prl30)
    left
    (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
   (roughing
    (tool tnmm-71 tmaxp-ptl90)
    left
    (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) ) ) )


(tup
 (roughing
  (tool dnmm-71 tmaxp-pdl93)
  left
  (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) )
1.0->
(skp 40 (tup))
```

anc-program:

```
(tup
 (roughing
  (tool dnmm-71 tmaxp-pdl93)
  left
  (geo (tup (p 70 40) (p 70 30) (p 150 30))) )
 (roughing
  (tool dnmm-71 tmaxp-pdl93)
  left
  (geo (tup (p 110 30) (p 110 25) (p 150 25))) )
 (roughing
  (tool dnmm-71 tmaxp-pdl93)
  left
  (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
 (roughing
  (tool dnmm-71 tmaxp-pdr93)
  right
  (geo (tup (p 110 25) (p 110 20) (p 120 20) (p 120 25))) )
 (roughing
  (tool dnmm-71 tmaxp-pdr93)
  right
  (geo (tup (p 0 25) (p 50 25) (p 60 40))) )
 (roughing
  (tool dnmm-71 tmaxp-pdr93)
  right
  (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) )
 (roughing
  (tool dnmm-71 tmaxp-pdl93)
  left
  (geo (tup (p 40 25) (p 40 20) (p 50 20) (p 50 25))) ) )
```

Acknowledgements

Besides the authors several other people have contributed, directly or indirectly, to the COLAB and/or μCAD2NC systems. In particular, our thanks go to Hans-Günter Hein, Klaus Elsbernd and Bernd Reuther. Also, we want to express our gratitude to Prof. Dr. Michael M. Richter for inspiring our AI involvement [redacted] paper.

References

[Baader and Hanschke, 1991a] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991. A long version is available as DFKI Research Report RR-91-10.

[Baader and Hanschke, 1991b] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. Research Report RR-91-10, DFKI GmbH, 1991.

[Boley *et al.*, 1991] H. Boley, P. Hanschke, K. Hinkelmann, and M. Meyer. COLAB: A Hybrid Knowledge Compilation Laboratory. 3rd International Workshop on Data, Expert Knowledge and Decisions: Using Knowledge to Transform Data into Information for Decision Support, September 1991.

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

[Helm and Marriott, 1986] Richard Helm and Kim Marriott. Declarative graphics. In E. Shapiro, editor, *Third International Conference on Logic Programming (ICLP)*, LNCS 225, pages 513–527, London, July 1986. Springer Verlag.

[Hinkelmann, 1991] Knut Hinkelmann. Forward logic evaluation: Developing a compiler from a partially evaluated meta interpreter. In W.-M. Lippe, editor, *Workshop Alternative Konzepte für Sprachen und Rechner*. Westfälische Wilhelms-Universität Münster, 1991.

[Klauck *et al.*, 1991] Christoph Klauck, Ralf Legleitner, and Ansgar Bernadi. FEAT-REP: Representing features in CAD/CAM. Technical report, 4th International Symposium on Artificial Intelligence: Applications in Informatics, Cancun, Mexiko, 1991.

[Kowalski, 1982] Robert Kowalski. Logic as a computer language for children. In *European Conference on Artificial Intelligence (ECAI)*, pages 2–10, 1982.

[Meyer and Jakfeld, 1991] M. Meyer and C. Jakfeld. How to use CONTAX – a Constraint System over Taxonomies. ARC-TEC Discussion Paper 91-04, DFKI GmbH, P. O. Box 2080, Kaiserslautern, Germany, March 1991.

[Pereira, 1986] F. Pereira. Can drawing be liberated from the von Neumann Style. In Michael van Caneghem and H.D. Warren, editors, *Logic Programming and its Applications*, volume 2 of *Ablex series in Artificial Intelligence*. 1986.

[Schmalhofer *et al.*, 1991] Franz Schmalhofer, Otto Kuehn, and Gabriele Schmidt. Integrated knowledge acquisition from text, previously solved cases, and expert memories. *Applied Artificial Intelligence*, 5:311–337, 1991.

A FORWARD Sources

A.1 Feature Aggregation

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;               microCAD2NC                     ;;;
;;;               FORWARD  part                   ;;;
;;;          Feature Aggregation Rules            ;;;
;;;   (c) Knut Hinkelmann                         ;;;
;;;       Martin Harm        September 1991       ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; Features derived by these rules are asserted into the ABox
; of TAXON. Attributes common to all features are the
; leftmost and rightmost surface the feature is covering.
; This is necessary to chaeck neighbarhood of surfaces and features.


; A shoulder is a feature consisting of two components:
; The ground is a longturning surface.
; The flank is either a descending surface (on the left)
; or an ascending surface (on the right) of the longturning
; surface. An additional condition is that the radius of the
; longturning surface is not greater than that of the
; descending or ascending surface, respectively.
;
; Examples:
;                           |     \
;                           | or   \
;        -------------------|       \-----------------
;

(up (add-data shoulder _featid
                        (tup (tup ground _ltsid)
                             (tup flank _id2)
                             (tup leftmost _l)
                             (tup rightmost _id2)))
    (asc-tc _id2  '(tup (tup center1 _zl)
                        (tup center2 _zr)
                        (tup radius1 _rad)
                        (tup radius2 _rado)))
    (neighbor _r _id2 )
    (data longturningsurface _ltsid
                              '(tup  (tup radius _rad-lts)
                                     (tup leftmost _l)
                                     (tup rightmost _r)))
    (>= _rad-lts _rad)
    (< _rad-lts _rado)
    (is _featid (make-instance-name shoulder _ltsid _id2)))
```



```
(up (add-data shoulder _featid
                        (tup (tup ground _ltsid)
                             (tup flank _id2)
                             (tup leftmost _id2)
                             (tup rightmost _r)))
    (desc-tc _id2 '(tup (tup center1 _zl)
                        (tup center2 _zl)
                        (tup radius1 _rado)
                        (tup radius2 _rad)))
    (neighbor _id2 _l)
    (data longturningsurface  _ltsid
                              '(tup (tup radius _rad-lts)
                                     (tup leftmost _l)
                                    (tup rightmost _r)))
    (>= _rad-lts _rad)
    (< _rad-lts _rado)
    (is _featid (make-instance-name shoulder _id2 _ltsid) ))


; A groove is an aggregation of two shoulder:
; a left shoulder and a right shoulder with common ground:
;
;          |                        /
;          |                       /
;          |______________________/
;
(up (add-data groove _featid
                      (tup (tup leftflank _id1)
                           (tup ground _id2)
                           (tup rightflank _id3)
                           (tup leftmost _lsleft)
                           (tup rightmost _rsright)))
    (data rshoulder _rshid
                    '(tup (tup ground _id2)
                          (tup flank _id3)
                          (tup leftmost _rsleft)
                          (tup rightmost _rsright)))
    (data lshoulder _lshid
                    '(tup (tup ground _id2)
                          (tup flank _id1)
                          (tup leftmost _lsleft)
                          (tup rightmost _lsright)))
    (is _featid (make-instance-name groove _id1 _id2 _id3)))


; There are three definitions for a longturning surface:
; 1. Each cylinder is a longturning surface
; 2. Starting from a descending surface a longturning surface
;    Extends to the right until either the workpiece ends or
```

```
;                     ns of a surface exceeds the radius of the descinding
;                     The radius of such a longturning surface is the
;                      the highest surface covered by the longturning surface.
;
;                     ____.........        (the dashed line shows the
;                        |____/\___         longturning surface)
;                                |
;
; 3.                  ıs 2. starting at an ascending surface and going
;                     ıft.

(up                   longturningsurface _cyl
                                          (tup (tup radius _rad)
                                               (tup leftmost _cyl)
                                               (tup rightmost _cyl)))
                      _cyl (tup (tup center1 _zl)
                                (tup center2 _zr)
                                (tup radius1 _rad)
                                (tup radius2 _rad))))

(up                   longturningsurface _featid
                                          (tup (tup radius _rad)
                                               (tup leftmost _l)
                                               (tup rightmost _rightm)))

                      ight-end
                      (tup (tup center1 _zl)
                           (tup center2 _zr)
                           (tup radius1 _rad-first)
                           (tup radius2 _rad-limit)))
                      _rightm _right-end)
                      once
                       '(sub-lts-from-left (tup rad-max _rad-first)
                                           (tup rad-limit _rad-limit)
                                           (tup leftmost _l)
                                           (tup rightmost _rightm))))
                      -tc _l _rightm)
                      d (make-instance-name lts _l _rightm)))

(ft                   rom-left
                      max _rad-max)       ;;; bisheriges maximum
                      limit _rad-limit)   ;;; obere schranke( darf nicht! erreicht werden)
                      most _l)
                      tmost _tc))
                      _next-tc _tc)
                      _next-tc
                      '(tup (tup center1 _zl)
                            (tup center2 _zr)
                            (tup radius1 _rada)
                            (tup radius2 _radb)))
```

```
                  '(tup leftmost _next-tc)))

t (sub-lts-from-right (tup rad-max _the-max)
                      (tup rad-limit _rad-limit)
                      (tup rightmost _tc)
                      (tup leftmost _tc))
  _the-max)

n (peak (feat _id1 _id2)
        (tup leftmost _id1)
        (tup rightmost _id2))
  (asc-tc _id1 '(tup (tup center1 _zl)
                     (tup center2 _zr1)
                     (tup radius1 _rada)
                     (tup radius2 _radb)))
  (neighbor _id1 _id2)
  (desc-tc _id2 '(tup (tup center1 _zr1)
                      (tup center2 _zr2)
                      (tup radius1 _radb)
                      (tup radius2 _radb2)))
  )

n (nutch (feat _id1 _id2)
         (tup leftmost _id1)
         (tup rightmost _id2))
  (desc-tc _id1 '(tup (tup center1 _zl)
                      (tup center2 _zr1)
                      (tup radius1 _rada)
                      (tup radius2 _radb)))
  (neighbor _id1 _id2)
  (asc-tc _id2 '(tup (tup center1 _zr1)
                     (tup center2 _zr2)
                     (tup radius1 _radb)
                     (tup radius2 _radb2))))

n (different-tc _tc1 _tc2)
  (nou _tc1 _tc2))

scending truncated cone: First radius is greater than second radius
l (asc-tc _x (tup (tup center1 _zl)
                  (tup center2 _zr)
                  (tup radius1 _rada)
                  (tup radius2 _radb)))
  (truncone _x '(tup (tup center1 _zl)
                     (tup center2 _zr)
                     (tup radius1 _rada)
                     (tup radius2 _radb)))
  (< _rada _radb))
```



```
; Descending truncated cone: First radius is smaller than second radius
(rl (desc-tc _x (tup (tup center1 _zl)
                     (tup center2 _zr)
                     (tup radius1 _rada)
                     (tup radius2 _radb)))
    (truncone _x '(tup (tup center1 _zl)
                       (tup center2 _zr)
                       (tup radius1 _rada)
                       (tup radius2 _radb)))
    (> _rada _radb))


; Truncated cone is the general form of a rotation symmetric surface.
; Therefore every surface is a truncated cone.
(rl (truncone _x (tup (tup center1 _zl)
                      (tup center2 _zr)
                      (tup radius1 _rada)
                      (tup radius2 _rada)))
    (cylinder _x '(tup (tup center1 _zl)
                       (tup center2 _zr)
                       (tup radius1 _rada)
                       (tup  radius2 _rada))))

(rl (truncone _x (tup (tup center1 _zl)
                      (tup center2 _zl)
                      (tup radius1 _rada)
                      (tup radius2 _radb)))
    (ring _x '(tup (tup center1 _zl)
                   (tup center2 _zl)
                   (tup radius1 _rada)
                   (tup radius2 _radb))))

(rl (truncone _x (tup (tup center1 _zl)
                      (tup center2 _zl)
                      (tup radius1 _rada)
                      (tup radius2 0)))
    (circle _x '(tup (tup center1 _zl)
                     (tup center2 _zl)
                     (tup radius1 _rada)
                     (tup radius2 0))))

(rl (truncone _x (tup (tup center1 _zl)
                      (tup center2 _zl)
                      (tup radius1 0)
                      (tup radius2 _radb)))
    (circle _x '(tup (tup center1 _zl)
                     (tup center2 _zl)
                     (tup radius1 0)
                     (tup radius2 _radb))))
```

```
(rl (truncone _x (tup (tup center1 _zl)
                      (tup center2 _zr)
                      (tup radius1 _rada)
                      (tup radius2 0)))
    (cone _x '(tup (tup center1 _zl)
                   (tup center2 _zr)
                   (tup radius1 _rada)
                   (tup radius2 0))))

(rl (truncone _x (tup (tup center1 _zl)
                      (tup center2 _zr)
                      (tup radius1 0)
                       (tup radius2 _radb)))
    (cone _x '(tup (tup center1 _zl)
                   (tup center2 _zr)
                   (tup radius1 0)
                   (tup radius2 _radb))))
```


A.2 TAXON Access Functions and Reasoning Strategies

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;               microCAD2NC                            ;;;
;;;              FORWARD  part                           ;;;
;;;    TAXON Access and Reasoning Strategies             ;;;
;;;    (c) Knut Hinkelmann                               ;;;
;;;        Thomas Labisch                                ;;;
;;;        Martin Harm        September 1991             ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; The definition of DATA retrieves attribute values of a concept's
; insatnces from the TAXON ABox. It applies the access function TX
; with specifier INSTANCE to get all instances of the given concept
; and retrieves all the attributes for this instance.

(hn (data _concept _instance  _attr-terms)
    (member _instance (tx instances _concept))
    (is _attr-terms (data-attributes _instance _attr-terms))
    ;(rf-terpri)
    ;(is _dummy (rf-terpri))
    ;(rf-print "Retrieval from TAXON ABox: ")
    ;(rf-pprint '(data _concept _instance  _attr-terms))
    )
;(hn (data _concept _instance  _attr-terms)
;;    (naf (member _instance (tx instances _concept)))
;    (add-data _concept _instance  _attr-terms)
;    (retain '(_concept _instance  _attr-terms)))


; The function DATA-ATTRIBUTES retrieves all attribute values
; for a given instance.

(ft (data-attributes _instance))
(ft (data-attributes _instance (tup (tup _attr _val)))
    (is _val (tx attr-filler _attr _instance))
    '(tup (tup _attr _val)))
(ft (data-attributes _instance (tup (tup _attr _val) (tup _attr2 _val2) | _attr-terms))
    (is _val (tx attr-filler _attr _instance))
    (is _rest-terms (data-attributes _instance
                                     '(tup (tup _attr2 _val2) | _attr-terms)))
    '(tup (tup _attr _val) | _rest-terms))
```

```
; RETAIN pushes derived facts onto the retain stack such that they can be
; applied to trigger forward rules.
; The argument of RETAIN can be the conclusion of a rule, which
; has to be asserted as an instance into the TAXON ABox. Such a
; conclusion has the form:
;     (ADD-DATA (<concept-name>
;                <instance-name>
;                (tup (<attr1> <val1>)
;                     ...
;                     (<attrN> <valN>))))

(hn (retain (add-data _concept _instance-name _args))
    (not-reached '(data _concept _instance-name _args))
    (is t (tx assert-ind? _instance-name _concept _args))
    (rf-terpri)
    (rf-pprint "Derived Fact asserted into TAXON ABox:")
    (rf-pprint '(_concept _instance-name _args))
    (push-fact-retain '(data _concept _instance-name _args)))
(hn (retain _Fact) (nou _Fact '(add-data _x _y _z))
                   (not-reached _Fact)
                   ;(rf-terpri)
                   ;(rf-pprint "Derived Fact:")
                   ;(rf-pprint _Fact)
                   (push-fact-retain _Fact))


; To trigger forward rules with facts, that are DATA-terms
; (i.e. instances from TAXON), the instance is realized in TAXON
; to find all concepts the instance belongs to (concept closure).
; Rules are triggered with each of these conept associations.

(hn (tx-unify _x _x))

(hn (open-node _Fact) (is _x (get-open-node))
                      (tx-unify '(data _old-concept _name  _attr-terms) _x)
                      (next-open-node)
                      (is _cc (tx concept-closure _name))
                      (member _concept _cc)
                      (is _Fact
                         '(data _concept _name  _attr-terms)))
(hn (open-node _Fact) (is _Fact (get-open-node))
                      (nou _Fact '(data _x _y _z))
                      (next-open-node))
(hn (open-node _Fact) (not-open-node-at-end) (open-node _Fact))


(hn (member _e (tup _e | _l)))
(hn (member _e (tup _a | _l)) (member _e _l))

(hn (nou _x _x) ! unknown)
```



```
(hn (nou _x _y))


; FORWARD-Reasoning Strategies:
;
; Bottom-up reasonig as applied in mCAD2NC is simulated in the
; RELFUN system. Therefor the bidirectional rules are transformed
; into RELFUN hn-rules and compiled into an extended RFM with
; special forward code area and retain stack.
; As reasoning strategies only breadt-first search is applied
; in our example.

; Breadth-first Search:
;
; (bf-all  '(tup _Fact | _Rest) _Inference) has as its value
; the list of consequences for its second argument derived
; in depth-first search.

(ft (bf-all (tup _Fact | _Rest) _Inference-pattern)
    (fc-initialize)
    (satisfied '(tup _Fact | _Rest))
    (bf-alist '(tup _Fact | _Rest) _Inference-pattern))
(hn (bf-all _Fact _Inference-pattern)
    (nou _Fact '(tup | _x))
    (fc-initialize)
    ;@_Fact
    (forward _Fact _Conclusion)
    false)
(ft (bf-all _Fact _Inference-pattern)
    (forward-all)
    (is _Inferences (collect-facts))
    (is _Inference-list (filter _Inference-pattern _Inferences))
    (reset-retain)
    _Inference-list)

(hn (bf-alist (tup _Fact | _Rest) _Inference-pattern)
    (forward _Fact _Inference)
    false)
(ft (bf-alist (tup _Fact | _Rest) _Inference-pattern)
    (bf-alist _Rest _Inference-pattern))

(hn (forward-all)
    (open-node _Fact)
    (forward _Fact _Conclusion)
    false)
(hn (forward-all))
```

```
; The initial facts of for forward reasoning must be satisfied, so that
; the can be used for proving premises of rules.
; Instead of simply testing it would be possible to assert them if
; they are not already satisfied.
(hn (satisfied (tup)))
(hn (satisfied (tup _Fact | _Rest))
    ;@_Fact
    (satisfied _Rest))


(hn (not-reached _Conclusion)
    (is t (subsumes-value _Conclusion)))
```


B TAXON Sources

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;            microCAD2NC                 ;;;
;;;            TAXON  part                 ;;;
;;;             Features                   ;;;
;;; (c) Philipp Hanschke September 1991 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;Intersting features defined in this file
(hierarchy      FEATURE
                COMPOSED ATOMIC
                ;DESCENDING ASCENDING
                ;HOLLOW FILLED
                RSHOULDER LSHOULDER
                LONGTURNINGSURFACE
                lts-from-right
                lts-from-left
                GROOVE
                TRUNCONE CYLINDER CONE RING CIRCLE
                ASC-CONE DESC-CONE
                ASC-RING DESC-RING
                ASC-tc DESC-tc
                DESC-CIRCLE  ASC-CIRCLE
                INSERTION SHOULDER)
;The concrete domain of rational numbers with comparison
;operators and boolean connectives is assigned to the tag RA.
(doma RA edom-real-ord)

;Some simple predicates of the concrete domain.
(pred  >0       (RA (x) (> x 0)))
(pred  <0       (RA (x) (< x 0)))
(pred  >=0      (RA (x) (>= x 0)))
(pred  <=0      (RA (x) (<= x 0)))
(pred  =0       (RA (x) (= x 0)))
(pred  <=       (RA (x y) (<= x y)))
(pred  >=       (RA (x y) (>= x y)))
(pred  <        (RA (x y) (< x y)))
(pred  >        (RA (x y) (> x y)))
(pred  !=       (RA (x y) (!= x y)))
(pred  =        (RA (x y) (= x y)))


;Seperate atomic and composed objects
(prim   atomic)
(conc   composed        (not atomic))
```

```
;A truncated cone is given by two centers and two radii.
;It should not degenerate.
(attr   center1
        center2
        radius1
        radius2)

#|(conc truncone
                (and atomic
                     (ra center1)
                     (ra center2)
                     (>=0 radius1)
                     (>=0 radius2)
                     (or (and (=  center1 center2)
                              (!= radius1 radius2))
                         (and (!= center1 center2)
                              (or (>0 radius1)
                                  (>0 radius2))))))
|#
(pred   tc-condition
                (ra (radius1 radius2 center1 center2)
                        (and    (>=0 radius1)
                                 (>=0 radius2)
                                (or (and (=  center1 center2)
                                         (!= radius1 radius2))
                                    (and (!= center1 center2)
                                         (or (>0 radius1)
                                             (>0 radius2)))))))

(conc   truncone
                (and atomic
                     (tc-condition radius1 radius2 center1 center2)))


;A ring is a very flat surface
(conc   ring
                (and truncone (=  center1 center2)))


;Two adjectives, filled and hollow, suitable for  non rings. They replace
;the in/out resp. left/right resp. +/- flags determining the orientation of the
;surface.
(conc   filled
                (and (< center1 center2)))
(conc   hollow  (and (> center1 center2)))

;two adjectives, ascending and descending, suitable for filled truncated cones.
(conc   ascending
                (and (<= radius1 radius2)))
(conc   descending
                (and (>= radius1 radius2)))
```



```
;A cylinder
(conc   cylinder
                (and truncone (=  radius1 radius2)))

;A circle is a ring where one radius is 0
(conc   circle
                (and ring (or (=0 radius1) (=0 radius2))))


;A cone
(conc   cone
                (and truncone (or (=0 radius1) (=0 radius2))))


;Applying some adjectives to ring, circle, and cone.
(conc   asc-tc
                (and truncone ascending))
(conc   desc-tc
                (and truncone descending))
(conc   asc-ring
                (and ring ascending))
(conc   desc-ring
                (and ring descending))
(conc   asc-circle
                (and ring (=0 radius1)))
(conc   desc-circle
                (and ring (=0 radius2)))
(conc   asc-cone
                (and truncone (=0 radius1)))
(conc   desc-cone
                (and truncone (=0 radius2)))


;Seperate shoulders from grooves
(prim   shoulder-class)
(conc    groove-class    (not shoulder-class))


;Neighbourhood is tested by the embedding system.
(prim   neighbouring)
(role   neighbours)


;The attributes leftmost and rightmost are used to check neighbourhood
;Till attributes agreements are available this must be done by the
;embedding system.
(attr leftmost rightmost)
(conc feature   (and (some leftmost truncone)
                     (some rightmost truncone)))

;left and right
(prim left)
```

```
(conc right (not left))

;For long turning surface only necessary conditions can be expressed
(prim   lts-sufficient)
(attr   radius)
(conc longturningsurface
                (and feature
                     (>=0 radius)
                     lts-sufficient))
(conc lts-from-right
                (and right longturningsurface))
(conc lts-from-left
                (and left longturningsurface))

;Attributes for long turning surfaces and shoulders
(attr   ground
        flank
        depth2width
        depth
        width
)


;the flank is left to the ground
(conc   flankleft2ground
                (and (<= (flank leftmost center1)
                         (ground leftmost center2))))
(conc   flankright2ground
                (and (>= (flank leftmost center1)
                         (ground leftmost center2))))

;There are two kinds of shoulders
(conc   shoulder-aux
                (and composed
                     shoulder-class
                     feature; that leftmost/rightmost are related to
                            ; leftmost/rightmost of the ground and the flank
                            ; has to be tested by the embedding system
                     neighbouring
                     (some ground longturningsurface)
                     (>0 depth)
                     ))
(conc   lshoulder
                (and shoulder-aux
                     flankleft2ground
                     (some flank descending)))
(conc   rshoulder
                (and shoulder-aux
                     flankright2ground
                      (some flank ascending)))
(conc   shoulder
                (and shoulder-aux
                     (or (and flankleft2ground
                              (some flank descending))
                         (and flankright2ground
                              (some flank ascending)))))


;A groove
(attr   leftflank
        rightflank)
(conc   groove
                (and feature
                     groove-class
                     (some leftflank descending)
                     (some rightflank ascending)
                     (some ground longturningsurface)
                     (>0 depth2width)
                     (>0 depth)
                     (>0 width)
                     neighbouring
                ))


(pred   insertion-condition
                (ra (d2w d) (and (< d2w  0.25) (< d 30))))
(conc   insertion
                (and groove (insertion-condition depth2width depth)))


;technological properties of a surface
(role   has-finish)
(attr   kind
        value)
(conc   finish
                (and (or (some kind top) (front-end::u-pred kind))
                     (or (some value top) (front-end::u-pred value))))
```

C RELFUN Sources

C.1 Library Functions

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;            microCAD2NC             ;;;
;;;            RELFUN part             ;;;
;;;          Library Functions         ;;;
;;; (c) Michael Sintek  September 1991 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(ft (tup) '(tup))
(ft (tup _x | _ r) '(tup _x | _r))

(ft (appfun (tup) _l) _l)
(ft (appfun (tup _h | _r) _l) (tup _h | (appfun _r _l)))

(ft (rf-reverse (tup)) '(tup))
(ft (rf-reverse (tup _h | _t))
  (appfun (rf-reverse _t) '(tup _h)))

(ft (rf-last (tup _last)) ! _last)
(ft (rf-last (tup _first | _rest)) (rf-last _rest))

(ft (rf-max (tup _x)) _x)
(ft (rf-max (tup _h | _t))
  (max _h (rf-max _t)))

(hn (rf-member _x (tup _x | _r)) !)
(hn (rf-member _x (tup _y | _r))
  (rf-member _x _r))

(ft (rf-nth 0 (tup _h | _t)) ! _h)
(ft (rf-nth _no (tup _h | _t))
  (rf-nth (1- _no) _t))
```


C.2 Transforming Aggregated Features to Classified Workpieces

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                         microCAD2NC                          ;;;
;;;                         RELFUN part                          ;;;
;;;  Transforming Aggregated Features to Classified Workpieces  ;;;
;;;           (c) Thomas Oltzen  September 1991                  ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(ft (feat2p _ret)
    (compacting _ret _x)
    (transform _x _x _z1)
    (sorting-in _z1 _z2)

    (sort-reverse _z2 _z3)
    (sort-kill-tomuch _z3 _z4)
    (sort-maxradius _z4 0 _maxradius)
    (sort-nft _z4 '(tup) _z5)
    (sort-height-adaptation _maxradius _z5 _zw1)
    (sort-2nftsintec _zw1 _z6)
    '(cwp _maxradius _z6))

;--------------------------------------------------------------

(hn (cy _x _o) (cylinder _x _o))
(hn (tr _x _o) (truncone _x _o))
(hn (rn _x _o) (ring _x _o))
(hn (ci _x _o) (circle _x _o))
(hn (co _x _o) (cone _x _o))

;--------------------------------------------------------------

(hn (compacting (tup) (tup)))
(hn (compacting (tup (data shoulder _fn
                                     (tup (tup ground _g)
                                          (tup flank _f)
                                          (tup leftmost _lm)
                                          (tup rightmost _rm)))
                      | _rest)
                 _new-ret)
    (compacting _rest _zw-erg)
    (is _new-ret '(tup (shoulder _fn
                                 (ground _g)
                                 (flank _f)
                                 (leftmost _lm)
                                 (rightmost _rm))
                       | _zw-erg)))
(hn (compacting (tup (data groove _fn
                                   (tup (tup leftflank _lf)
                                        (tup ground _g)
                                        (tup rightflank _rf)
```

```
                              (tup leftmost _lm)
                              (tup rightmost _rm)))
     | _rest)
  _new-ret)
g _rest _zw-erg)
t '(tup (groove _fn
                (leftflank _lf)
                (ground _g)
                (rightflank _rf)
                (leftmost _lm)
                (rightmost _rm))

 | _zw-erg)))
g (tup (data longturningsurface _fn
                               (tup (tup radius _rad)
                                    (tup leftmost _lm)
                                    (tup rightmost _rm)))
     | _rest)
  _new-ret)
g _rest _zw-erg)
st '(tup (longturningsurface _fn
                (radius _rad)
                (leftmost _lm)
                (rightmost _rm))
        | _zw-erg)))

g (tup _feat-kann-weg | _rest) _new-ret)
g _rest _new-ret))

alisator _z1 _z2 _r1 _r _r _z2))
alisator _z1 _z2 _r _r2 _r _z1))
alisator _z1 _z2 _r1 _r2 _r _zout)
)

)
(- _z2 (* (- _z2 _z1) (/ (- _r1 _r2)(- _r _r2))))))
alisator _z1 _z2 _r1 _r2 _r _zout)
)

)
(+ _z1 (* (- _z2 _z1) (/ (- _r2 _r1)(- _r _r1))))))
alisator _z1 _z2 _r1 _r2 _r _z2)
warnig-lts-has-a-not-good-radius)
2))
alisator _z1 _z2 _r1 _r2 _r _z1)
warnig-lts-has-a-not-good-radius)
))
 ls rs lmz rmz _r (p z1 r1) (p z2 r2) (p z3 r3)

 _ret
 (shoulder _fid (ground _lts)
          (flank _lms)
```



```
                    (leftmost _lms)
                    (rightmost _rms))
            (tup leftshoulder _lms _rms _lmz _rmz _r
                 (p _lmz _r1)(p _z2 _r)(p _rmz _r)))
    (leftmost-s _lms _lmz)
    (rightmost-s _rms _rmz)
    (tc _lms _lmz1 _lmz2 _lmsr1 _lmsr2)
    (is _r1 (maximal _lmsr1 _lmsr2))
    (searchid _lts _ret _o)
    (is _ov '(longturningsurface _lts (radius _r) (leftmost _lts-lm) _e3))
    (is _ov _o)
    (sort-normalisator _lmz1 _lmz2 _lmsr1 _lmsr2 _r _z2))

(hn (transfeat _ret
            (shoulder _fid (ground _lts)
                    (flank _rms)
                    (leftmost _lms)
                    (rightmost _rms))
            ( tup rightshoulder _lms _rms _lmz _rmz _r
                   (p _lmz _ )(p _z2 _r)(p _rmz _r1)))
    (leftmost-s _lms _lmz)
    (rightmost-s _rms _rmz)
    (tc _rms _lmsz1 _lmsz2 _lmsr1 _lmsr2)
    (is _r1 (maximal _lmsr1 _lmsr2))
    (searchid _lts _ret _o)
    (is _ov '(longturningsurface _lts (radius _r) _e3 (rightmost _lts-rm)))
    (is _ov _o)
    (sort-normalisator _lmsz1 _lmsz2 _lmsr1 _lmsr2 _r _z2))

(hn (transfeat _ret
            (groove _fid
                  (leftflank _lf)
                  (ground _lts)
                  (rightflank _sf)
                  (leftmost _lms)
                  (rightmost _rms))
            (tup groove _lms _rms _lmz _rmz _r
                 (p _lmz _rl)(p _ltslmz _r)(p _ltsrmz _r)(p _rmz _rr)))
    (leftmost-s _lms _lmz)
    (rightmost-s _rms _rmz)
    (tc _lms _e1 _e2 _lmsr1 _lmsr2)
    (is _rl (maximal _lmsr1 _lmsr2))
    (tc _rms _e3 _e4 _rmsr1 _rmsr2)
    (is _rr (maximal _rmsr1 _rmsr2))
    (searchid _lts _ret _o)
    (is _ov '(longturningsurface _lts (radius _r) (leftmost _lts-lm)
                        (rightmost _lts-rm)))
    (is _ov _o)
    (sort-normalisator _e1 _e2 _lmsr1 _lmsr2 _r _ltslmz)
    (sort-normalisator _e3 _e4 _rmsr1 _rmsr2 _r _ltsrmz))

(hn (leftmost-s _idf _lm)
```

```
    (tc _idf _c1 _c2 _r1 _r2)
    (is _lm (minimal _c1 _c2)))

(hn (rightmost-s _idf _lm)
    (tc _idf _c1 _c2 _r1 _r2)
    (is _lm (maximal _c1 _c2)))

(ft (maximal _a _b)
    (>= _a _b)
    _a)

(ft (maximal _a _b)
    (>= _b _a)
    _b)

(ft (minimal _a _b)
    (>= _a _b)
    _b)

(ft (minimal _a _b)
    (>= _b _a)
    _a)


(ft (tc _fid _c1 _c2 _r1 _r2)
    (tr _fid '(tup (tup center1 _c1) (tup center2 _c2 )
                    (tup radius1 _r1) (tup radius2 _r2))))
(ft (tc _fid _c1 _c2 _r1 _r2)
    (rn _fid '(tup (tup center1 _c1) (tup center2 _c2 )
                    (tup radius1 _r1) (tup radius2 _r2))))
(ft (tc _fid _c1 _c2 _r1 _r2)
    (cy _fid '(tup (tup center1 _c1) (tup center2 _c2 )
                    (tup radius1 _r1) (tup radius2 _r2))))
(ft (tc _fid _c1 _c2 _r1 _r2)
    (co _fid '(tup (tup center1 _c1) (tup center2 _c2 )
                    (tup radius1 _r1) (tup radius2 _r2))))
(ft (tc _fid _c1 _c2 _r1 _r2)
    (ci _fid '(tup (tup center1 _c1) (tup center2 _c2 )
                    (tup radius1 _r1) (tup radius2 _r2))))


(hn (transform _ret (tup) (tup)))
(hn (transform _ret (tup _f | _r) _out)
    (transfeat _ret _f _nf)
    (transform _ret _r _o)
    (is _out '(tup _nf | _o)))
(hn (transform _ret (tup _f | _r) _o)
    (transform _ret _r _o))


(hn (searchid _fid (tup _rf | _rr) _o)
    (is _rf '(longturningsurface _fid _r _lm _rm))
```



```
    (is _o '(longturningsurface _fid _r _lm _rm)))

(hn (searchid _fid (tup _rf | _r) _o)
    (searchid _fid _r _o))


(hn (sorting-in (tup) (tup)))
(hn (sorting-in (tup _f | _r) _o)
    (sorting-in _r _s)
    (sort-feat _f _s _o))
```

```
(hn (sort-feat _data-feat (tup) (tup _data-feat)))
(hn (sort-feat _data-feat _s-feat (tup _data-feat | _s-feat))
    (tup-car _s-feat _car)
    (sort-kleiner? _data-feat _car))
(hn (sort-feat _data-feat (tup _f | _r) _out)
    (is _out '(tup _f | _o))
    (sort-feat _data-feat _r _o))

(hn (sort-kleiner? (tup _n1 _ls1 _rs1 | _r1)
                   (tup _n2 _ls2 _rs2 | _r2))
    (sort-id-topo _rs1 _rs2)
    (sort-id-topo _ls2 _ls1))

(hn (sort-id-topo _n1 _n1))
(hn (sort-id-topo _n1 _n2)
    (neighbor _n1 _n2))
(hn (sort-id-topo _n1 _n2)
    (neighbor _n1 _a)
    (sort-id-topo _a _n2))

(hn (tup-car (tup _x | _y) _x))
(hn (tup-cdr (tup _x | _y) _y))

(hn (sort-rev (tup) _revl _revl))
(hn (sort-rev (tup _f | _r) _revl _o)
    (sort-rev _r '(tup _f | _revl) _o))

(hn (sort-reverse _i _o)
    (sort-rev _i '(tup) _o))

(hn (sort-kill-tomuch (tup) (tup)))
(hn (sort-kill-tomuch (tup _f | _r) _o)
    (sort-kill-lower _f _r _ze1)
    (sort-kill-tomuch _ze1 _ze)
    (is _o '(tup _f | _ze)))

(hn (sort-kill-lower _ele (tup) (tup)))
(hn (sort-kill-lower _ele (tup _f | _r) _out)
    (sort-lower _f _ele)
    !
    (sort-kill-lower _ele _r _out))
(hn (sort-kill-lower _ele (tup _f | _r) _out)
    (sort-kill-lower _ele _r _ze)
    (is _out '(tup _f | _ze)))


;; is _ele1 lower as _ele2 ?
(hn (sort-lower _ele1 _ele1))
(hn (sort-lower (tup _fname _e1 _e2 _lmz _rmz _r | _fplist)
                (tup _fname2 _e3 _e4 _lmz2 _rmz2 _r2 | _fplist2))

    (>= _lmz _lmz2)
```



```
    (<= _rmz _rmz2)
    (sort-2tup-lower _fplist _fplist2))

; constraint is _z1 < _z2
(hn (sort-point-lower (p _z1 _r) (p _z2 _r) (p _fz _fr))
    (>= _fr _r)
    (< _z1 _z2)
    (>= _fz _z1)
    (>= _z2 _fz))
(hn (sort-point-lower (p _z1 _r1)(p _z2 _r2) (p _z1 _r1)))
(hn (sort-point-lower (p _z1 _r1)(p _z2 _r2) (p _z2 _r2)))
(hn (sort-point-lower (p _z1 _r1)(p _z2 _r2) (p _fz _fr))
    (> _r1 _r2)
    (< _z1 _z2)
    (>= _fz _z1)
    (>= _z2 _fz)
    (> _fr _r2)
    (is _a (- _z2 _z1))
    (is _b (- _r1 _r2))
    (is _d (- _z2 _fz))
    (is _c (- _fr _r2))
    (<= (/ _d _c) (/ _a _b)))
(hn (sort-point-lower (p _z1 _r1)(p _z2 _r2) (p _fz _fr))
    (< _r1 _r2)
    (< _z1 _z2)
    (>= _fz _z1)
    (>= _z2 _fz)
    (> _fr _r1)
    (is _a (- _z2 _z1))
    (is _b (- _r2 _r1))
    (is _d (- _z2 _fz))
    (is _c (- _fr _r1))
    (<= (/ _d _c) (/ _a _b)))
```

```
(hn (sort-tuppoint-lower _fp (tup _p1 _p2 | _r))
    (sort-point-lower _p1 _p2 _fp))
(hn (sort-tuppoint-lower _fp (tup _p1 | _r))
    (sort-tuppoint-lower _fp _r))

(hn (sort-2tup-lower (tup) _r))
(hn (sort-2tup-lower (tup _fp | _rfp) _r)
    (sort-tuppoint-lower _fp _r)
    (sort-2tup-lower _rfp _r))

(hn (sort-nft (tup) _nft-tup _nft-tup))
(hn (sort-nft (tup _f | _r) _nft-tup _out)
    (sort-insert-nft _f _nft-tup _zw-erg)
    (sort-nft _r _zw-erg _out))

(hn (sort-insert-nft _feat (tup) (tup (nft _feat (tup)))))
(hn (sort-insert-nft _feat
                     (tup (nft _nft-feat _tuppel) | _rest-nft)
                     _out-tup)
    (sort-kleiner? _feat _nft-feat)
    (sort-insert-nft _feat _tuppel _zw-erg)
    (is _zw-erg2 '(nft _nft-feat _zw-erg))
    (is _out-tup '(tup _zw-erg2 | _rest-nft)))
(hn (sort-insert-nft _feat
                     (tup _nft-list | _rest-nft)
                     _out-nft)
    (sort-insert-nft _feat _rest-nft _zw-erg)
    (is _out-nft '(tup _nft-list | _zw-erg)))

(hn (sort-2nftsintec (tup) (tup)))
(hn (sort-2nftsintec (tup _nft-feat | _rest) _out)
    (sort-2sintec-nft _nft-feat _zw-erg)
    (sort-2nftsintec _rest _zw-erg2)
    (is _out '(tup _zw-erg | _zw-erg2)))

(hn (sort-height-adaptation _maxrad (tup) (tup)))
(hn (sort-height-adaptation _maxrad (tup _nft-feat | _rest)
                            (tup _new-nft | _new-rest))
    (sort-height-adaptation-nft _maxrad _nft-feat _new-nft)
    (sort-height-adaptation _maxrad _rest _new-rest))

(hn (sort-height-adaptation-nft _maxrad (nft _feat _tupel)
                                (nft _newfeat _new-tupel))
    (sort-height-adaptation-radius _maxrad _feat _zw-feat)
    (sort-height-adaptation-radius _maxrad _zw-feat _newfeat)
    (sort-height-minradius _feat _minrad)
    (sort-height-adaptation _minrad _tupel _new-tupel))

(hn (sort-height-adaptation-radius _maxradius
                                   (tup groove _e1 _e2 _e3 _e4 _e5
                                        (p _z1 _r1)(p _z2 _r2)
```



```
                                        (p _z3 _r3)(p _z4 _r4))
                                   (tup groove _e1 _e2 _e3 _e4 _e5
                                        (p _nz1 _maxradius)(p _z2 _r2)
                                        (p _z3 _r3)(p _z4 _r4)))
    (< _maxradius _r1)
    (sort-normalisator _z1 _z2 _r1 _r2 _maxradius _nz1))
(hn (sort-height-adaptation-radius _maxradius
                                   (tup groove _e1 _e2 _e3 _e4 _e5
                                        (p _z1 _r1)(p _z2 _r2)
                                        (p _z3 _r3)(p _z4 _r4))
                                   (tup groove _e1 _e2 _e3 _e4 _e5
                                        (p _z1 _r1)(p _z2 _r2)
                                        (p _z3 _r3)(p _nz4 _maxradius)))
    (< _maxradius _r4)
    (sort-normalisator _z3 _z4 _r3 _r4 _maxradius _nz4))
(hn (sort-height-adaptation-radius _maxradius
                                   _feat
                                   _feat))

(hn (sort-height-minradius (tup _fn _e1 _e2 _e3 _e4 _e5 | _plist) _minrad)
    (sort-height-minradius-plist _plist 10000000 _minrad))

(hn (sort-height-minradius-plist (tup) _x _x))
(hn (sort-height-minradius-plist (tup (p _z _r) | _rest) _rad _minrad)
    (sort-height-minradius-plist _rest (minimal _r _rad) _minrad))

(hn (sort-2sintec-nft (nft (tup leftshoulder _w1 _w2 _w3 _w4 _w5 _p1 _p2 _p3)
                           _rest)
                      (nft _feat-out _rest-out))
    (is _feat-out '(lsh (flk (tup _p1 _p2))
                        (grd (tup _p2 _p3))))
    (sort-2nftsintec _rest _rest-out))
(hn (sort-2sintec-nft (nft (tup rightshoulder _w1 _w2 _w3 _w4 _w5 _p1 _p2 _p3)
                           _rest)
                      (nft _feat-out _rest-out))
    (is _feat-out '(rsh (grd (tup _p1 _p2))
                        (flk (tup _p2 _p3))))
    (sort-2nftsintec _rest _rest-out))
(hn (sort-2sintec-nft (nft (tup groove _w1 _w2 _w3 _w4 _w5
                                _p1 _p2 _p3 _p4)
                           _rest)
                      (nft _feat-out _rest-out))
    (is _feat-out '(grv (flk (tup _p1 _p2))
                        (grd (tup _p2 _p3))
                        (flk (tup _p3 _p4))))
    (sort-2nftsintec _rest _rest-out))

(hn (sort-maxradius (tup) _r _r))
(hn (sort-maxradius (tup (tup leftshoulder _w1 _w2 _w3 _w4 _w5
                              (p _w6 _r1)(p _w7 _r2)(p _w8 _r3))
                         | _rest)
                    _r
```

```
              _ro t)
    (sort-maxradius _rest
              (maximal (maximal (maximal _r _r1)
                                 _r2)
                        _r3)
              _rout))
(hn (sort-maxradius (tup (tup rightshoulder _w1 _w2 _w3 _w4 _w5
                              (p _w6 _r1)(p _w7 _r2)(p _w8 _r3))
                         | _rest)
              _r
              _rout)
    (sort-maxradius _rest
              (maximal (maximal (maximal _r _r1)
                                 _r2)
                        _r3)
              _rout))
(hn (sort-maxradius (tup (tup groove _w1 _w2 _w3 _w4 _w5
                              (p _w6 _r1)(p _w7 _r2)(p _w8 _r3)(p _w9 _r4))
                         | _rest)
              _r
              _rout)
    (sort-maxradius _rest
              (maximal (maximal (maximal (maximal _r _r1)
                                          _r2)
                                 _r3)
                        _r4)
              _rout))
```


C.3 Recursive Workpiece Classification

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;              microCAD2NC              ;;;
;;;              RELFUN part              ;;;
;;; Recursive Workpiece Classification ;;;
;;; (c) Michael Sintek     August 1991 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; The program mainly uses RELFUN functions defined by ft clauses which
; are all deterministic

; classify wp
; -----------

(ft (class-feat _wp-rng) ; _wp-rng must be in rng notation
  (is _rad (max-rad _wp-rng))
  (is _class-feat (class-sec _rad _wp-rng))
  '(cwp _rad _class-feat))


; classify section (class-sec <max> <wp-rng>)
; -------------------------------------------

; end at (tup) or (tup _x)
(ft (class-sec _max (tup)) !
  '(tup))

(ft (class-sec _max (tup _x)) ! ; _x thrown out because a single supporting
  '(tup))                       ; point cannot become a feature

; ignore max height cylinders
(ft (class-sec _max (tup (rng _z1 _ri _max) (rng _z2 _max _ro) | _rest)) !
  (class-sec _max '(tup (rng _z2 _max _ro) | _rest)))

; groove: begins with max and split-wp-rng succeeds
(ft (class-sec _max (tup (rng _z _max _ro) | _rest))
  (is (tup _left-wp-rng | _rest-wp-rng)    ; check for and return part
          (split-wp-rng _max _rest)) !     ; up to maximum
  ;(rf-print groove)
  ;(rf-print '(tup (rng _z _max _ro) | _rest))
  ;(rf-print '(tup (rng _z _max _ro) | _left-wp-rng))
  (tup (class-grv '(tup (rng _z _max _ro) | _left-wp-rng))
       | (class-sec _max _rest-wp-rng)))

; right shoulder: doesn't start with max and split-wp-rng succeeds
(ft (class-sec _max (tup (rng _z _ri _ro) | _rest))
  (/= _ri _max)
  (is (tup _left-wp-rng | _rest-wp-rng)
      (split-wp-rng _max '(tup (rng _z _ri _ro) | _rest))) !
  ;(rf-print right-shoulder)
  ;(rf-print '(tup (rng _z _max _ro) | _rest))
```

```
(class-sec _max _rest-wp-rng)))

p1) (find-flank _wp-rng 0))
p2) (find-flank (reverse-wp-rng _rest-wp1) 0))
min) (rf-last _test-flank1))
k1-min (max-rad _rest-wp2)))
d-flank _wp-rng _ground-max))
ind-flank (reverse-wp-rng _rest) _ground-max))
und) _ground-max _grd _refined-ground)
g _r-flank))
 (flk _r-flank-rev)) _refined-ground))

) (find-flank _wp-rng 0))
in) (rf-last _test-flank))
k-min (max-rad _rest-wp)))
d-flank _wp-rng _ground-max))
_grd _refined-ground)
refined-ground))

verse-wp-rng _wp-rng))))

(grd (tup)) (tup)) !)

ax (grd (tup)) (tup)) !)

x
```



```
        (grd (tup
                (rng _fz _ground-max _ground-max)
                (rng _lz _ground-max _ground-max)))
        _refined-ground)
  (is _refined-ground (class-sec _ground-max _ground))
  (is (tup (rng _fz _fri _fro) | _rest-ground) _ground)
  (is (rng _lz _lri _lro) (rf-last _ground)))

; find flank (find-flank <wp-rng> <min>)
; -------------------------------------

; (a) asc. ring
(ft (find-flank (tup (rng _z _ri _ro) | _rest) _min)
  ;(rf-print a)
  (> _ro _ri) !
  '(tup (tup (rng _z _ri _ri)) | (tup (rng _z _ro _ro) | _rest)))

; (b) desc. ring passing or downto min
(ft (find-flank (tup (rng _z _ri _ro) | _rest) _min)
  ;(rf-print b)
  (>= _min _ro ) !
  '(tup (tup (rng _z _ri _min)) | (tup (rng _z _min _ro) | _rest)))

; (c) desc. last ring above min
(ft (find-flank (tup (rng _z _ri _ro)) _min) !
  ;(rf-print c)
  ; (>= _ri _ro) (> _ro _min) holds
  '(tup (tup (rng _z _ri _ro)) | (tup)))

; (d) asc. truncone
(ft (find-flank (tup (rng _z1 _ri1 _ro1) (rng _z2 _ri2 _ro2) | _rest) _min)
  ;(rf-print d)
  (> _ri2 _ro1) ! ; (>= _ri1 _ro1) (> _ro1 _min) holds
  '(tup (tup (rng _z1 _ri1 _ro1))
       | (tup (rng _z1 _ro1 _ro1) (rng _z2 _ri2 _ro2) | _rest)))

; (e1) desc. truncone downto min
(ft (find-flank (tup (rng _z1 _ri1 _ro1) (rng _z2 _ri2 _ro2) | _rest) _min)
  ;(rf-print e1)
  (= _min _ri2) ! ; (>= _ri1 _ro1) (> _ro1 _min) holds
  '(tup (tup (rng _z1 _ri1 _ro1) (rng _z2 _min _min))
       | (tup (rng _z2 _ri2 _ro2) | _rest)))

; (e2) desc. truncone passing min
(ft (find-flank (tup (rng _z1 _ri1 _ro1) (rng _z2 _ri2 _ro2) | _rest) _min)
  ;(rf-print e2)
  (> _min _ri2) ! ; (>= _ri1 _ro1) (> _ro1 _min) holds
  (is _z12 (split-truncone _z1 _ro1 _z2 _ri2 _min))
  '(tup (tup (rng _z1 _ri1 _ro1) (rng _z12 _min _min))
       | (tup (rng _z12 _min _min) (rng _z2 _ri2 _ro2) | _rest)))

; (f) cylinder or desc. truncone above min
```

```
(ft (find-flank (tup (rng _z1 _ri1 _ro1) (rng _z2 _ri2 _ro2) | _rest) _min)
  ;(rf-print f)
  ; fall through case: (>= _ri1 _ro1 _ri2) holds
  (is (tup _rest-flank | _rest-wp)
      (find-flank '(tup (rng _z2 _ri2 _ro2) | _rest) _min))
  '(tup (tup (rng _z1 _ri1 _ro1) | _rest-flank) | _rest-wp))


; reverse feature & reverse feature list
; ---------------------------------------

(ft (reverse-feature (nft (lsh (flk _flank) (grd _ground)) _refined-ground))
  (is _flank-rev (reverse-wp-rng _flank))
  (is _ground-rev (reverse-wp-rng _ground))
  (is _refined-ground-rev (reverse-feature-list _refined-ground))
  '(nft (rsh (grd _ground-rev) (flk _flank-rev)) _refined-ground-rev))
```



```
(ft (reverse-feature (nft (rsh (grd _ground) (flk _flank)) _refined-ground))
  (is _flank-rev (reverse-wp-rng _flank))
  (is _ground-rev (reverse-wp-rng _ground))
  (is _refined-ground-rev (reverse-feature-list _refined-ground))
  '(nft (lsh (flk _flank-rev) (grd _ground-rev)) _refined-ground-rev))

(ft (reverse-feature (nft (grv (flk _flank1) (grd _ground) (flk _flank2))
                          _refined-ground))
  (is _flank1-rev (reverse-wp-rng _flank1))
  (is _ground-rev (reverse-wp-rng _ground))
  (is _flank2-rev (reverse-wp-rng _flank2))
  (is _refined-ground-rev (reverse-feature-list _refined-ground))
  '(nft (grv (flk _flank2-rev) (grd _ground-rev) (flk _flank1-rev))
        _refined-ground-rev))

(ft (reverse-feature-list (tup))
  '(tup))
(ft (reverse-feature-list (tup _h | _t))
  (appfun (reverse-feature-list _t) (tup (reverse-feature _h))))

; auxiliary functions
; -------------------

(ft (split-wp-rng _max (tup (rng _z _max _ro) | _rest)) !
  '(tup (tup (rng _z _max _max)) | (tup (rng _z _max _ro) | _rest)))
(ft (split-wp-rng _max (tup (rng _z _ri _max) | _rest)) !
  '(tup (tup (rng _z _ri _max)) | (tup (rng _z _max _max) | _rest)))
(ft (split-wp-rng _max (tup (rng _z _ri _ro) | _rest))
  (is (tup _left-wp-rng | _rest-wp-rng) (split-wp-rng _max _rest))
  '(tup (tup (rng _z _ri _ro) | _left-wp-rng) | _rest-wp-rng))

(ft (split-truncone _z1 _r1 _z2 _r2 _min)
  (is _h (- _r1 _r2))
  (+ _z1 (/ (* (- _z2 _z1) (- _h (- _min _r2))) _h)))

(ft (max-rad (tup)) ! 0)
(ft (max-rad (tup (rng _z _ri _ro) | _rest-rng-list))
  (max _ri _ro (max-rad _rest-rng-list)))

(ft (reverse-wp-rng (tup)) '(tup))
(ft (reverse-wp-rng (tup (rng _z _ri _ro) | _rest))
  (is _mirr-z (- 0 _z))
  (appfun (reverse-wp-rng _rest) '(tup (rng _mirr-z _ro _ri))))
```

C.4 Transforming CWP's from rng-Notation to p-Notation

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                    microCAD2NC                     ;;;
;;;                    RELFUN part                     ;;;
;;; Transforming CWP's from rng-Notation to p-Notation ;;;
;;; (c) Michael Sintek                  September 1991 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(ft (cwp-rng2p (cwp _globals _class-feat-rng))
  (is _class-feat-p (rng2p-feature-list _class-feat-rng))
  '(cwp _globals _class-feat-p))


(ft (rng2p-feature (nft (lsh (flk _flank) (grd _ground)) _refined-ground))
  (is _flank-p (rng2p _flank))
  (is _ground-p (rng2p _ground))
  (is _refined-ground-p (rng2p-feature-list _refined-ground))
  '(nft (lsh (flk _flank-p) (grd _ground-p)) _refined-ground-p))

(ft (rng2p-feature (nft (rsh (grd _ground) (flk _flank)) _refined-ground))
  (is _flank-p (rng2p _flank))
  (is _ground-p (rng2p _ground))
  (is _refined-ground-p (rng2p-feature-list _refined-ground))
  '(nft (rsh (grd _ground-p) (flk _flank-p)) _refined-ground-p))

(ft (rng2p-feature (nft (grv (flk _flank1) (grd _ground) (flk _flank2))
                        _refined-ground))
  (is _flank1-p (rng2p _flank1))
  (is _ground-p (rng2p _ground))
  (is _flank2-p (rng2p _flank2))
  (is _refined-ground-p (rng2p-feature-list _refined-ground))
  '(nft (grv (flk _flank1-p) (grd _ground-p) (flk _flank2-p))
        _refined-ground-p))


(ft (rng2p-feature-list (tup))
  '(tup))
(ft (rng2p-feature-list (tup _h | _t))
  (tup (rng2p-feature _h) | (rng2p-feature-list _t)))


(ft (rng2p (tup)) ! '(tup))
(ft (rng2p (tup (rng _z _ri _ro) | _rest))
  (/= _ri _ro) !
  (tup '(p _z _ri) | (rng2p '(tup (rng _z _ro _ro) | _rest))))
(ft (rng2p (tup (rng _z _ro _ro) | _rest))
  (tup '(p _z _ro) | (rng2p _rest)))
```


C.5 Skeletal Plans

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;            microCAD2NC            ;;;
;;;            RELFUN part            ;;;
;;;           Skeletal Plans          ;;;
;;; (c) Michael Sintek  September 1991 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


#|

Input: classified workpieces
----------------------------


class-wp ::= (cwp <globals> <feature-list>)

globals ::= "set of global annotations; momentarily the max radius;
             should additionally contain the material and the
             surface quality"

feature-list ::= (tup { <nested-feature> }* )

nested-feature ::= (nft <feature> <feature-list>)

feature ::= <leftshoulder> | <rightshoulder> | <groove> | ...

leftshoulder ::= (lsh <flank> <ground>)

rightshoulder :: (rsh <ground> <flank>)

groove ::= (grv <flank> <ground> <flank>)

flank ::= (flk <coordinates>)

ground ::= (grd <coordinates>)

coordinates ::= (tup { <point> }* )

point ::= (p <num> <num>)  ; z-coordinate and radius
```

```
etal plans
-----------

an ::= (skp <globals> <sac-plan>)
       ; sac = sequential/alternative/commutative

(tup)                  ; empty plan
action>                ; atomar action
seq <plan-list>)       ; sequential
com <plan-list>)       ; commutative
alt <plan-list>)       ; alternative

= (tup { <plan> }* )

roughing> | <finishing> | ...

 (roughing <tool> <way> <geometry>)

ol <plate> <holder>)

 | right ; means: working into this direction

 (geo <coordinates>)



skeletal plan
=============

l-plan (cwp _globals _features))
skp/com _features))
als _skp))



 (tup)) !


 (tup _h | _t))
skp/cfeat _h) (skp/com _t)))


complex features like nft = nested feature)
------------------------------------------
```



```
(ft (skp/cfeat (nft _feat _refined-feat))
  (norm-seq (skp/feat _feat) (skp/com _refined-feat)))

; additional complex/nested features ...


; skp/feat (for unnested features like lsh, rsh, grv)
; ---------------------------------------------------

(ft (skp/feat (lsh (flk _flank) (grd _ground))) !
  (skp/lsh _flank _ground))

(ft (skp/feat (rsh (grd _ground) (flk _flank))) !
  (skp/rsh _ground _flank))

(ft (skp/feat (grv (flk _flank1) (grd _ground) (flk _flank2)))
  (skp/grv _flank1 _ground _flank2))


; skeletal plans for unnested features
; ------------------------------------

; global constants for demo version:

(ft (wp-material) high-alloy-steel)

(ft (quality) normal)


; skp/lsh
; -------

(ft (skp/lsh _flank _ground)
  (is _fl-gr (append-coord _flank _ground))
  (tool-sel roughing (wp-material) (quality)
            0 (rf-max (compute-flank-angles _flank))
            left _tools)
  (check-alternatives
   (gen-roughing-alternatives _tools left '(geo _fl-gr))))


; skp/rsh
; -------

(ft (skp/rsh _ground _flank)
  (is _gr-fl (append-coord _ground _flank))
  (tool-sel roughing (wp-material) (quality)
            0 (rf-max (compute-flank-angles (reverse-coord _flank)))
            right
            _tools)
```

```
  (check-alternatives
   (gen-roughing-alternatives _tools right '(geo _gr-fl))))


; skp/grv
; -------

(ft (skp/grv _flank1 _ground _flank2)
  (is _fl-gr-fl (append-coord _flank1 (append-coord _ground _flank2)))
  (is _max1 (rf-max (compute-flank-angles _flank1)))
  (is _max2 (rf-max (compute-flank-angles (reverse-coord _flank2))))
  (tool-sel roughing (wp-material) (quality)
            _max1 _max2 right _tools1)
  (tool-sel roughing (wp-material) (quality)
            _max2 _max1 left _tools2)
  (is _right-alt (check-alternatives (gen-roughing-alternatives
                                      _tools1 right '(geo _fl-gr-fl))))
  (is _left-alt (check-alternatives (gen-roughing-alternatives
                                     _tools2 left '(geo _fl-gr-fl))))
  (norm-alt (norm-seq _left-alt _right-alt)
            (norm-seq _right-alt _left-alt)))


; additional skeletal plans ...
```



```
; tool-sel (tool selection)
; -------------------------

; format:
; (tool-sel <process> <wp-material> <quality> <alpha> <beta> <way>
;           (tup { (tup <plate> <holder>) }+ ))

; place precomputed tool selections here ...


; CONTAX tool selection:

(hn (tool-sel _process _wp-material _quality _alpha _beta _direction
              _tools)

  (rf-print contax\ tool\ selection\:)
  (rf-print \ \ arguments\:\ )
  (rf-princ _process) (rf-princ \ )
  (rf-princ _wp-material) (rf-princ \ )
  (rf-princ _quality) (rf-princ \ )
  (rf-princ _alpha) (rf-princ \ )
  (rf-princ _beta) (rf-princ \ )
  (rf-princ _direction)
  (rf-terpri)
  (rf-print \ \ propagating...)
  (is _tools (get-cn-tools
              (cn global hard
                  ; (tup quality _quality)
                  (tup wp-material _wp-material)
                  (tup process _process)
                  (tup alpha _alpha)
                  (tup beta _beta)
                  (tup direction _direction)
                  (tup tool lathe-tools)
                  (tup holder holders)
                  (tup plate plate-geometries)
                  (tup edge-angle edge-angles)
                  (tup tc-edge-angle tc-edge-angles))
              tool holder))
  (rf-terpri)
  (rf-princ \ \ results\:\ )
  (rf-princ _tools)
  (rf-terpri) (rf-terpri))
```

```
; init routine for tool selection via CONTAX
; must be run prior to calling (skeletal-plan ...)

(hn (init-tool-sel)

  (rf-print contax\ initialization\:)

  (rf-print \ \ creating\ constraint\ variables...)
  ; (cn cv quality hrcl (tup qualities))
  (cn cv wp-material hrcl (tup wp-materials))
  (cn cv process hrcl (tup processes))
  (cn cv tool hrcl (tup lathe-tools))
  (cn cv holder hrcl (tup holders))
  (cn cv alpha hrcl (tup acute-angles))
  (cn cv beta hrcl (tup acute-angles))
  (cn cv direction hrcl (tup directions))
  (cn cv plate hrcl (tup plate-geometries))
  (cn cv edge-angle hrcl (tup edge-angles))
  (cn cv tc-edge-angle hrcl (tup tc-edge-angles))

  (rf-print \ \ creating\ constraint\ instances...)
  (cn ci inst-ho-to hard holder-tool holder tool)
  (cn ci inst-pr-ho hard process-holder process holder)
  (cn ci inst-ho-de1 hard holder-desc1 holder direction)
  (cn ci inst-ho-de2 hard holder-desc2 holder tc-edge-angle)
  (cn ci inst-ho-de3 hard holder-desc3 holder plate)
  (cn ci inst-pr-ma-to hard process-material-tool
        process wp-material tool)
  (cn ci inst-pl-ea hard plate-eangle plate
        edge-angle)
  (cn ci inst-pr-ea hard process-eangle process edge-angle)
  ; (cn ci inst-tc-ea-al hard tc-ea-al tc-edge-angle
         ; edge-angle alpha)
  (cn ci inst-tc-beta hard tc-beta tc-edge-angle beta)

  (rf-print ok.)
  (rf-terpri))
```



```
; append-coord
; ------------

(ft (append-coord (tup) _f) !
  _f)
(ft (append-coord _f (tup)) !
  _f)
(ft (append-coord (tup (p _z _r)) (tup (p _z _r) | _rest)) !
  '(tup (p _z _r) | _rest))
(ft (append-coord (tup _h | _t) _r)
  (tup _h | (append-coord _t _r)))


; reverse-coord
; -------------

(ft (reverse-coord (tup)) ! '(tup))
(ft (reverse-coord (tup (p _z _r) | _rest))
  (is _mirr-z (- 0 _z))
  (appfun (reverse-coord _rest) '(tup (p _mirr-z _r))))


; compute-flank-angles (only for descending flanks)
; -------------------------------------------------

(ft (compute-flank-angles (tup)) !
  '(tup))

(ft (compute-flank-angles (tup (p _z _r))) !
  '(tup))

(ft (compute-flank-angles (tup (p _z _r1) (p _z _r2) | _rest)) ! ; 90 deg ring
  (tup 90
      | (compute-flank-angles '(tup (p _z _r2) | _rest))))

(ft (compute-flank-angles (tup (p _z1 _r1) (p _z2 _r2) | _rest))
  (tup (rad2rounded-deg (atan (/ (- _r1 _r2) (- _z2 _z1))))
      | (compute-flank-angles '(tup (p _z2 _r2) | _rest))))

(ft (rad2rounded-deg _rad)
  (* (round (/ (+ (* _rad 57.295779513082323) 2.9999) 3)) 3))


; gen-roughing-alternatives
; -------------------------

(ft (gen-roughing-alternatives (tup) _way _geo)
  '(tup))
(ft (gen-roughing-alternatives
      (tup (tup _tool _holder) | _rest) _way _geo)
  (norm-alt '(roughing (tool _tool _holder) _way _geo)
```

```
          (gen-roughing-alternatives _rest _way _geo)))


; check-alternatives
; ------------------

(ft (check-alternatives (tup)) !
  (rf-print tool\ selection\ unsuccessfull))
(ft (check-alternatives _x) _x)


; get-cn-tools
; ------------

; extract a list of tools/holders from the return value
; of the CONTAX propagation

(ft (get-cn-tools (tup _vars | _values) _tool-var-name _holder-var-name)
  (is _tool-pos (find-pos _tool-var-name _vars))
  (is _holder-pos (find-pos _holder-var-name _vars))
  (get-cn-tools1 _values _tool-pos _holder-pos))

(ft (get-cn-tools1 (tup) _tool-pos _holder-pos) !
  '(tup))

(ft (get-cn-tools1 (tup _h | _t) _tool-pos _holder-pos)
  (add-if-new (tup (rf-nth _tool-pos _h)
                   (rf-nth _holder-pos _h))
              (get-cn-tools1 _t _tool-pos _holder-pos)))
```



```
; 2. skp access/transform functions
; ================================

; a) get-skp-top
; --------------

(ft (get-skp-top (skp _globals _sac-plan))
  (skp-top/plan _sac-plan))


(ft (skp-top/plan (tup)) !
  '(tup))

(ft (skp-top/plan (com _com)) !
  (skp-top/com _com))

(ft (skp-top/plan (seq _seq)) !
  (skp-top/seq _seq))

(ft (skp-top/plan (alt _alt)) !
  (skp-top/alt _alt))

(ft (skp-top/plan _action)
  _action)


(ft (skp-top/com (tup)) !
  '(tup))

(ft (skp-top/com (tup _h | _t))
  (norm-com (skp-top/plan _h) (skp-top/com _t)))


(ft (skp-top/seq (tup)) !
  '(tup))

(ft (skp-top/seq (tup _h | _t))
  (skp-top/plan _h))


(ft (skp-top/alt (tup)) !
  '(tup))

(ft (skp-top/alt (tup _h | _t))
  (norm-alt (skp-top/plan _h) (skp-top/alt _t)))


; b) execute-skp-action
; ---------------------

; skp-exec-action (skp-exec-action <skp> <actspec>)
```

```
;    -> (tup <action-list> <rest-plan> <costs>)
; -----------------------------------------------

(ft (skp-exec-action (skp _globals _sac-plan) _actspec)
  (is (tup _actions _rest-plan _costs) (skp-exec/plan _sac-plan _actspec))
  '(tup _actions (skp _globals _rest-plan) _costs))


(ft (skp-exec/plan (tup) _actspec) !
  '(tup (tup) (tup) 0))

(ft (skp-exec/plan (com _com) _actspec) !
  (skp-exec/com _com _actspec))

(ft (skp-exec/plan (seq _seq) _actspec) !
  (skp-exec/seq _seq _actspec))

(ft (skp-exec/plan (alt _alt) _actspec) !
  (skp-exec/alt _alt _alt _actspec))

(ft (skp-exec/plan _action _actspec)
  (skp-exec/act _action _actspec))


(ft (skp-exec/com (tup) _actspec) !
  '(tup (tup) (tup) 0))

(ft (skp-exec/com (tup _h | _t) _actspec)
  (is (tup _h-actions _h-rest-plan _h-costs) (skp-exec/plan _h _actspec))
  (is (tup _t-actions _t-rest-plan _t-costs) (skp-exec/com _t _actspec))
  (tup (appfun _h-actions _t-actions)
       (norm-com _h-rest-plan _t-rest-plan)
       (+ _h-costs _t-costs)))
```



```
(ft (skp-exec/seq (tup) _actspec) !
  '(tup (tup) (tup) 0))

(ft (skp-exec/seq (tup _h | _t) _actspec)
  (skp-exec/seq1 (skp-exec/plan _h _actspec) _t _actspec))

(ft (skp-exec/seq1 (tup _h-actions (tup) _h-costs) _t _actspec) !
  (is (tup _t-actions _t-rest-plan _t-costs) (skp-exec/com _t _actspec))
  (tup (appfun _h-actions _t-actions)
       _t-rest-plan
       (+ _h-costs _t-costs)))

(ft (skp-exec/seq1 (tup _h-actions _h-rest-plan _h-costs) _t _actspec)
  (tup _h-actions
       (norm-seq _h-rest-plan (norm-rest-seq _t))
       _h-costs))


(ft (skp-exec/alt _alt (tup) _actspec) ! ; no more alternatives to check
  '(tup (tup) (alt _alt) 0))

(ft (skp-exec/alt _alt (tup _h | _t) _actspec)
  (skp-exec/alt1 _alt (skp-exec/plan _h _actspec) _h _t _actspec))

(ft (skp-exec/alt1 _alt (tup _h-actions _h _costs) _h _t _actspec) !
  (skp-exec/alt _alt _t _actspec)) ; try remaining alternatives

(ft (skp-exec/alt1 _alt (tup _h-actions _h-rest-plan _h-costs) _h _t _actspec)
  '(tup _h-actions _h-rest-plan _h-costs)) ; alternative found


(ft (skp-exec/act (roughing _tool _way _geo) _tool) !
  '(tup (tup (roughing _tool _way _geo)) (tup) 1)) ; costs: 1

(ft (skp-exec/act _action _actspec)
  '(tup (tup) _action 0))
```

```
; (c) extract-actions (extract-actions <sac-skp>)
; -----------------------------------------------

(ft (extract-actions (tup)) !
  '(tup))

(ft (extract-actions (com _com)) !
  (extract-actions/list _com))

(ft (extract-actions (seq _seq)) !
  (extract-actions/list _seq))

(ft (extract-actions (alt _alt)) !
  (extract-actions/list _alt))

(ft (extract-actions _action)
  '(tup _action))


(ft (extract-actions/list (tup)) !
  '(tup))

(ft (extract-actions/list (tup _h | _t))
  (appfun (extract-actions _h) (extract-actions/list _t)))


; (d) get-tools (from action-list; removing duplicates)
; -----------------------------------------------------

(ft (get-tools (tup)) ! '(tup))
(ft (get-tools (tup (roughing _tool _way _geo) | _rest))
  (is _rest-tools (get-tools _rest))
  (add-if-new _tool _rest-tools))
```



```
; (e) count-com-actions
; ---------------------

(ft (count-com-actions (skp _globals _sac-plan))
  (cca/plan _sac-plan))


(ft (cca/plan (tup)) !
  0)

(ft (cca/plan (com _com)) !
  (cca/com _com))

(ft (cca/plan (seq (tup _h | _t))) !
  (cca/plan _h))

(ft (cca/plan (alt (tup _h | _t))) ! ; only consider first alternative!
  (cca/plan _h))

(ft (cca/plan _action)
  1)


(ft (cca/com (tup)) !
  0)

(ft (cca/com (tup _h | _t))
  (+ (cca/plan _h) (cca/com _t)))
```

```
; 3. skp normalizations
; ====================

; norm-com (norm-com <plan1> <plan2>)
; ----------------------------------

(ft (norm-com (tup) _plan) !
  _plan)

(ft (norm-com _plan (tup)) !
  _plan)

(ft (norm-com (com _com1) (com _com2)) !
  (is _com12 (appfun _com1 _com2))
  '(com _com12))

(ft (norm-com _plan (com _com)) !
  '(com (tup _plan | _com)))

(ft (norm-com (com _com) _plan) !
  (is _cp (appfun _com '(tup _plan)))
  '(com _cp))

(ft (norm-com _plan1 _plan2)
  '(com (tup _plan1 _plan2)))


; norm-seq (norm-seq <plan1> <plan2>) & (norm-rest-seq <tup>)
; ----------------------------------------------------------

(ft (norm-seq (tup) _plan) !
  _plan)

(ft (norm-seq _plan (tup)) !
  _plan)

(ft (norm-seq (seq _seq1) (seq _seq2)) !
  (is _seq12 (appfun _seq1 _seq2))
  '(seq _seq12))

(ft (norm-seq _plan (seq _seq)) !
  '(seq (tup _plan | _seq)))

(ft (norm-seq (seq _seq) _plan) !
  (is _sp (appfun _seq '(tup _plan)))
  '(seq _sp))

(ft (norm-seq _plan1 _plan2)
  '(seq (tup _plan1 _plan2)))
```



```
; (norm-rest-seq <tup>) : <tup> is expected to be the rest of a
; normalized sequence!

(ft (norm-rest-seq (tup)) !
  '(tup))

(ft (norm-rest-seq (tup _plan)) !
  _plan)

(ft (norm-rest-seq _tup)
  '(seq _tup))


; norm-alt (norm-alt <plan1> <plan2>)
; ----------------------------------

(ft (norm-alt (tup) _plan) !
  _plan)

(ft (norm-alt _plan (tup)) !
  _plan)

(ft (norm-alt (alt _alt1) (alt _alt2)) !
  (is _alt12 (appfun _alt1 _alt2))
  '(alt _alt12))

(ft (norm-alt _plan (alt _alt)) !
  '(alt (tup _plan | _alt)))

(ft (norm-alt (alt _alt) _plan) !
  (is _ap (appfun _alt '(tup _plan)))
  '(alt _ap))

(ft (norm-alt _plan1 _plan2)
  '(alt (tup _plan1 _plan2)))
```

```
;;;;;;;;;
      ;;;
      ;;;
      ;;;
 1991 ;;;
;;;;;;;;;

 }* )

tup))) !

<p
et-skp-top _skp))) _skp
ctions _best-rest-skp)

plan _best-rest-skp)))
```

```
; get-promising-tool
; ------------------

; call: (get-promising-tool
;        <best-tool-so-far> <best-costs-so-far>
;        <best-actions-so-far> <best-rest-skp-so-far>
;        <rest-tools> <skp>
;        <best-tool> <best-costs> <best-actions> <best-rest-skp>)

(hn (get-promising-tool
     _best-tool-so-far _best-costs-so-far
     _best-actions-so-far _best-rest-skp-so-far
     (tup) _skp                              ; no more tools
     _best-tool-so-far _best-costs-so-far
     _best-actions-so-far _best-rest-skp-so-far) !)

(hn (get-promising-tool
     _best-tool-so-far _best-costs-so-far
     _best-actions-so-far _best-rest-skp-so-far
     (tup _tool | _rest-tools) _skp
     _best-tool _best-costs _best-actions _best-rest-skp)
  (once (compute-costs _skp _tool _actions _rest-plan _costs))
  (> _costs _best-costs-so-far) !
  (get-promising-tool2 ; check if rest plan is empty!
   _tool _costs
   _actions _rest-plan
   _rest-tools _skp
   _best-tool _best-costs _best-actions _best-rest-skp))

(hn (get-promising-tool
     _best-tool-so-far _best-costs-so-far
     _best-actions-so-far _best-rest-skp-so-far
     (tup _tool | _rest-tools) _skp
     _best-tool _best-costs _best-actions _best-rest-skp)
  (get-promising-tool
   _best-tool-so-far _best-costs-so-far
   _best-actions-so-far _best-rest-skp-so-far
   _rest-tools _skp
   _best-tool _best-costs _best-actions _best-rest-skp))


(hn (get-promising-tool2
     _best-tool-so-far _best-costs-so-far
     _best-actions-so-far (tup) ; empty rest plan ->
     _rest-tools _skp           ; ignore rest tools !
     _best-tool-so-far _best-costs-so-far
     _best-actions-so-far (tup)) !)

(hn (get-promising-tool2
     _best-tool-so-far _best-costs-so-far
     _best-actions-so-far _best-rest-skp-so-far
```



```
      _rest-tools _skp
      _best-tool _best-costs _best-actions _best-rest-skp)
  (get-promising-tool
   _best-tool-so-far _best-costs-so-far
   _best-actions-so-far _best-rest-skp-so-far
   _rest-tools _skp
   _best-tool _best-costs _best-actions _best-rest-skp))


(hn (compute-costs _skp _tool _actions _rest-plan _costs)
  (is (tup _actions _rest-plan _step-costs)
      (skp-exec-action _skp _tool))
  (is _costs (+ _step-costs (* 0.7 (count-com-actions _rest-plan)))))
```

C.7 Demo

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;            microCAD2NC            ;;;
;;;            RELFUN part            ;;;
;;;               Demo                ;;;
;;; (c) Michael Sintek  September 1991 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(ft (show _headline _x)
  (rf-terpri)
  (rf-print _headline)
  (rf-terpri)
  (rf-terpri)
  (rf-pprint _x)
  (rf-terpri)
  (rf-terpri)
  _x)


(ft (headline _headline _x)
  (rf-terpri)
  (rf-print _headline)
  (rf-terpri)
  (rf-terpri)
  _x)


(ft (cwp2anc _cwp)
 (headline anc-program\:
  (gen-anc-plan
   (headline qualitative\ simulation\:
    (show skeletal\ plan\:
          (skeletal-plan (show classified\ workpiece\: _cwp)))))))

(ft (rng2cwp _wp-rng)
  (cwp-rng2p
   (show classified\ workpiece\ in\ p-notation\:
 (class-feat (show workpiece\ in\ rng-notation\: _wp-rng)))))

(ft (tec2cwp)
  (feat2p (show features\: (bf-all '(truncone _a _b) _c))))


(ft (demo/r) ; pure relfun + contax demo
  (cwp2anc (rng2cwp (exa))))

(ft (demo/f) ; forward + taxon + relfun + contax demo
  (cwp2anc (tec2cwp)))
```



```
; example
; -------

#|
|                       D  E
40                      +--+
|                      /   |
35                    /    |
|                    /     |               F
30                  /      +---------------+
|  A            B  /                       |  G            H
25--------------+  + C                     |  +------------+
|               |  |                       |  |            |
20              +--+                       +--+            |
|                                                          |
15                                                         |
|                                                          |
10                                                         |
|                                                          |
5                                                          |
|                                                          |
0_____________________________________________________________
    10  20  30  40  50  60  70  80  90  100 110 120 130 140 150
|#

(ft (exa)
  '(tup (rng   0  0 25) ; A
        (rng  40 25 20) ; B
        (rng  50 20 25) ; C
        (rng  60 40 40) ; D
        (rng  70 40 30) ; E
        (rng 110 30 20) ; F
        (rng 120 20 25) ; G
        (rng 150 25  0))) ; H
```

D CONTAX Sources

D.1 Tools Database

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;            microCAD2NC           ;;;
;;;             CONTAX part          ;;;
;;;           Tools Database         ;;;
;;; (c) Joerg Mueller  September 1991 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;; Part 1: domain definitions

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          definitions of domains for tools (cutting plates)          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(dd lathe-tools ( finishing-tools roughturn-tools ))


(dd roughturn-tools ( universal-tools mm71 nma mm41 ))


(dd finishing-tools ( universal-tools mm53 cma ))


(dd universal-tools ( nmg mm52 RCMX ))


(dd mm71 ( DNMM-71 TNMM-71 SNMM-71 CNMM-71 ))


(dd mm41 ( TNMM-41 SNMM-41 DNMM-41 CNMM-41 ))


;;(dd mm53 ( TCMM-53 DCMM-53 SCMM-53 CCMM-53 VBMM-53 ))
;;(dd mm53 ( TCMM-53 DCMM-53 SCMM-53 CCMM-53 ))


;;(dd cma ( TCMA DCMA SCMA CCMA ))


(dd nma ( TNMA DNMA SNMA CNMA ))


(dd nmg (RNMG TNMG SNMG DNMG CNMG ))
```



```
(dd mm52 ( TCMM-52 DCMM-52 SCMM-52 CCMM-52 RCMM-52 ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;        domain definitions for tool systems (holders)                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; the names of the holders result from a projection of the relevant criteria
;; from the ISO names of workpieces.
;; For example TMAXP-PTL90 means:
;;             Holder type = TMAX-P
;;             fixing system = P
;;             form of cutting plate = T
;;             cutting direction = L (from right to left)
;;             tool cutting edge angle = 90


(dd holders ( tmax-p tmax-u ))

;; TMAX-P holders


(dd tmax-p ( pt ps pc pr pd ))


(dd pt      (
              TMAXP-PTL90
              TMAXP-PTL80
              TMAXP-PTL45
              TMAXP-PTN60
              TMAXP-PTR90
              TMAXP-PTR80
              TMAXP-PTR45
            ))
```

```
(dd ps     (
              TMAXP-PSL75
              TMAXP-PSL45
              TMAXP-PSN45
              TMAXP-PSR75
              TMAXP-PSR45
           ))

(dd pc     (
              TMAXP-PCL95
              TMAXP-PCL75
              TMAXP-PCN65
              TMAXP-PCR95
              TMAXP-PCR75
              TMAXP-PCR65
           ))

(dd pr     (
              TMAXP-PRL30
              TMAXP-PRL40
;;              TMAXP-RN
              TMAXP-PRR30
          ))

(dd pd     (
              TMAXP-PDL93
              TMAXP-PDR93
))

;; TMAX-U holders

;;(dd tmax-u ( st ss sv sr TMAXU-SCN95 TMAXU-SDL93 ))
#|

(dd tmax-u ( st ss sr TMAXU-SCN95 TMAXU-SDL93 ))

(dd st    (
             TMAXU-STL90
             TMAXU-STL75
             TMAXU-STN60
             TMAXU-STN45
             TMAXU-STR90
             TMAXU-STR75
          ))

(dd ss    (
```



```
              TMAXU-SSR75
              TMAXU-SSN45
              TMAXU-SSL75
              TMAXU-SSL45
           ))

(dd sv     (
              TMAXU-SVL93
              TMAXU-SVN72
              TMAXU-SVR93
           ))

(dd sr     (
              TMAXU-SRN
              TMAXU-SRN30
           ))
|#
;; definition of work processes)

(dd processes (roughing finishing))

;; definition of required workpiece qualities

(dd qualities ( normal critical ))

;; domain definitions for workpiece materials

(dd wp-materials ( steel cast alu ))

(dd steel ( building-steel alloy-steel stainless-steel ))

(dd cast ( GG GGG ))

(dd alloy-steel ( low-alloy-steel high-alloy-steel ))

;; domain definitions of relevant angles

(dd acute-angles (0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57
                  60 63 66 69 72 75 78 81 84 87 90))

;; definition of tool cutting edge angles

(dd tc-edge-angles ( 0 small-tc-angles big-tc-angles ))
```

```
(dd small-tc-angles ( 45 60 72 75 ))

(dd big-tc-angles ( 90 93 95 ))

;; edge-angle of the cutting plate


(dd edge-angles ( 0 small-edge-angles medium-edge-angles big-edge-angles ))

(dd small-edge-angles (0 35 ))

(dd medium-edge-angles (55 60))

(dd big-edge-angles (0 80 90 ))

;; domain definitions of cutting-plate geometry


;;(dd plate-geometries ( R S T D V C K ))
(dd plate-geometries ( R S T D C K ))

;; definition of cutting directions, which can be to left or to right


(dd directions ( left right ))
```



```
;; Part 2: constraint definitions

;; holder-tool describes the cutting-plates fitting to several holders for
;; reasons of geometry


(pc holder-tool (ho to) (holders lathe-tools)
          (pt TNMM-71)
          (pt TNMM-41)
          (pt TNMG)

          (ps SNMM-71)
          (ps SNMM-41)
          (ps SNMG)

          (pc CNMM-71)
          (pc CNMM-41)
          (pc CNMG)
          (pc CNMA)

          (pr RCMX)
          (pr RNMG)

          (pd DNMG)
          (pd DNMM-71)
          (pd DNMM-41)
#|
          (st TCMM-53)
          (st TCMM-52)
          (st TCMA)

          (ss SCMM-53)
          (ss SCMM-52)
          (ss SCMA)

          (sr RCMM-52)
          (sv VBMM-53)

          (TMAXU-SCN95 CCMM-52)
          (TMAXU-SCN95 CCMM-53)
          (TMAXU-SCN95 CCMA)

          (TMAXU-SDL93 DCMM-52)
          (TMAXU-SDL93 DCMM-53)
          (TMAXU-SDL93 DCMA )
|#
)


;; process-holder describes the fact that TMAX-P holders are favourable for
;; roughing, whereas TMAX-U holders should be preferred for finishing. If
```

```
;; quality restrictions are not so critical, for finishing, TMAX-S holders
;; may be used.


(pc process-holder (pr ho) (processes holders)
          (roughing tmax-p)
          (finishing tmax-u ))

;;

(pc holder-desc1 (ho di ) (holders directions)
          (TMAXP-PTL90 left )
          (TMAXP-PTR90 right )
          (TMAXP-PTL80 left )
          (TMAXP-PTR80 right )
          (TMAXP-PTN60 directions )
          (TMAXP-PTL45 left )
          (TMAXP-PTR45 right )

          (TMAXP-PSL75 left )
          (TMAXP-PSR75 right )
          (TMAXP-PSL45 left )
          (TMAXP-PSR45 right )
          (TMAXP-PSN45 directions )

          (TMAXP-PCL95 left )
          (TMAXP-PCR95 right )
          (TMAXP-PCL75 left )
          (TMAXP-PCR75 right )
          (TMAXP-PCR65 right )
          (TMAXP-PCN65 directions )

          (TMAXP-PDL93 left )
          (TMAXP-PDR93 right )

          (TMAXP-PRL30 left )
          (TMAXP-PRR30 right)
          (TMAXP-PRL40 left )
;;          (TMAXP-RN    directions )
#|
          (TMAXU-STL90 left 90 T)
          (TMAXU-STR90 right 90 T)
          (TMAXU-STL75 left 75 T)
          (TMAXU-STR75 right 75 T)
          (TMAXU-STN60 directions 60 T)
          (TMAXU-STN45 directions 45 T)

          (TMAXU-SSL75 left 75 S)
          (TMAXU-SSR75 right 75 S)
          (TMAXU-SSL45 left 45 S)
          (TMAXU-SSN45 directions 45 S)
          (TMAXU-SCN95 directions 95 C)
```



```
          (TMAXU-SDL93 left 93 D)
          (TMAXU-SVL93 left 93 V)
          (TMAXU-SVR93 right 93 V)
          (TMAXU-SVN72 directions 72 V)
          (TMAXU-SRN   directions 0 R)
          (TMAXU-SRN30 directions 0 R )
|#
)

(pc holder-desc2 (ho tc) (holders tc-edge-angles )
          (TMAXP-PTL90 90 )
          (TMAXP-PTR90 90 )
          (TMAXP-PTL80 80 )
          (TMAXP-PTR80 80 )
          (TMAXP-PTN60 60 )
          (TMAXP-PTL45 45 )
          (TMAXP-PTR45 45 )

          (TMAXP-PSL75 75 )
          (TMAXP-PSR75 75 )
          (TMAXP-PSL45 45 )
          (TMAXP-PSR45 45 )
          (TMAXP-PSN45 45 )

          (TMAXP-PCL95 95 )
          (TMAXP-PCR95 95 )
          (TMAXP-PCL75 75 )
          (TMAXP-PCR75 75 )
          (TMAXP-PCR65 65 )
          (TMAXP-PCN65 65 )

          (TMAXP-PDL93 93 )
          (TMAXP-PDR93 93 )

          (TMAXP-PRL30  0 )
          (TMAXP-PRR30  0 )
          (TMAXP-PRL40  0 )
;;          (TMAXP-RN     0 )
#|
          (TMAXU-STL90 left 90 T)
          (TMAXU-STR90 right 90 T)
          (TMAXU-STL75 left 75 T)
          (TMAXU-STR75 right 75 T)
          (TMAXU-STN60 directions 60 T)
          (TMAXU-STN45 directions 45 T)

          (TMAXU-SSL75 left 75 S)
          (TMAXU-SSR75 right 75 S)
          (TMAXU-SSL45 left 45 S)
          (TMAXU-SSN45 directions 45 S)
          (TMAXU-SCN95 directions 95 C)
          (TMAXU-SDL93 left 93 D)
```

```
        (TMAXU-SVL93 left 93 V)
        (TMAXU-SVR93 right 93 V)
        (TMAXU-SVN72 directions 72 V)
        (TMAXU-SRN   directions 0 R)
        (TMAXU-SRN30 directions 0 R )
|#
)

(pc holder-desc3 (ho pl) (holders plate-geometries)
        (TMAXP-PTL90  T)
        (TMAXP-PTR90  T)
        (TMAXP-PTL80  T)
        (TMAXP-PTR80  T)
        (TMAXP-PTN60  T)
        (TMAXP-PTL45  T)
        (TMAXP-PTR45  T)

        (TMAXP-PSL75  S)
        (TMAXP-PSR75  S)
        (TMAXP-PSL45  S)
        (TMAXP-PSR45  S)
        (TMAXP-PSN45  S)

        (TMAXP-PCL95  C)
        (TMAXP-PCR95  C)
        (TMAXP-PCL75  C)
        (TMAXP-PCR75  C)
        (TMAXP-PCR65  C)
        (TMAXP-PCN65  C)

        (TMAXP-PDL93  D)
        (TMAXP-PDR93  D)

        (TMAXP-PRL30 R)
        (TMAXP-PRR30  R)
        (TMAXP-PRL40  R)
;;        (TMAXP-RN     R)
#|
        (TMAXU-STL90 left 90 T)
        (TMAXU-STR90 right 90 T)
        (TMAXU-STL75 left 75 T)
        (TMAXU-STR75 right 75 T)
        (TMAXU-STN60 directions 60 T)
        (TMAXU-STN45 directions 45 T)

        (TMAXU-SSL75 left 75 S)
        (TMAXU-SSR75 right 75 S)
        (TMAXU-SSL45 left 45 S)
        (TMAXU-SSN45 directions 45 S)
        (TMAXU-SCN95 directions 95 C)
        (TMAXU-SDL93 left 93 D)
        (TMAXU-SVL93 left 93 V)
```



```
        (TMA U-SVR93 right 93 V)
        (TMA U-SVN72 directions 72 V)
        (TMA U-SRN   directions 0 R)
        (TMA U-SRN30 directions 0 R )
|#
)
```

```
;; process-material-tool specifies the usability of cutting-plates w.r.t. the
;; working-process to be done and the properties of materials.
;; The constraint reflects the suitability of the cutting-plate materials
;; (which are implicitly contained in their names) for certain workpiece
;; materials, e.g. short-cutting, long-cutting, stainless, warmfest, ???-hard,hard.


(pc process-material-tool (pr ma to) (processes wp-materials lathe-tools)
          (roughing  steel           mm71 )
          (roughing  cast            mm71 )
          (roughing  cast            nma )
          (roughing  stainless-steel mm41 )
          (roughing  alloy-steel     nma )
          (roughing  low-alloy-steel mm52 )
;;          (finishing low-alloy-steel mm53 )
;;          (finishing steel           mm52 )
;;          (finishing cast            mm52 )
;;          (finishing cast            cma )
          (processes alu             nmg )
          (processes steel           RCMX )
          (processes cast            RCMX ))
```

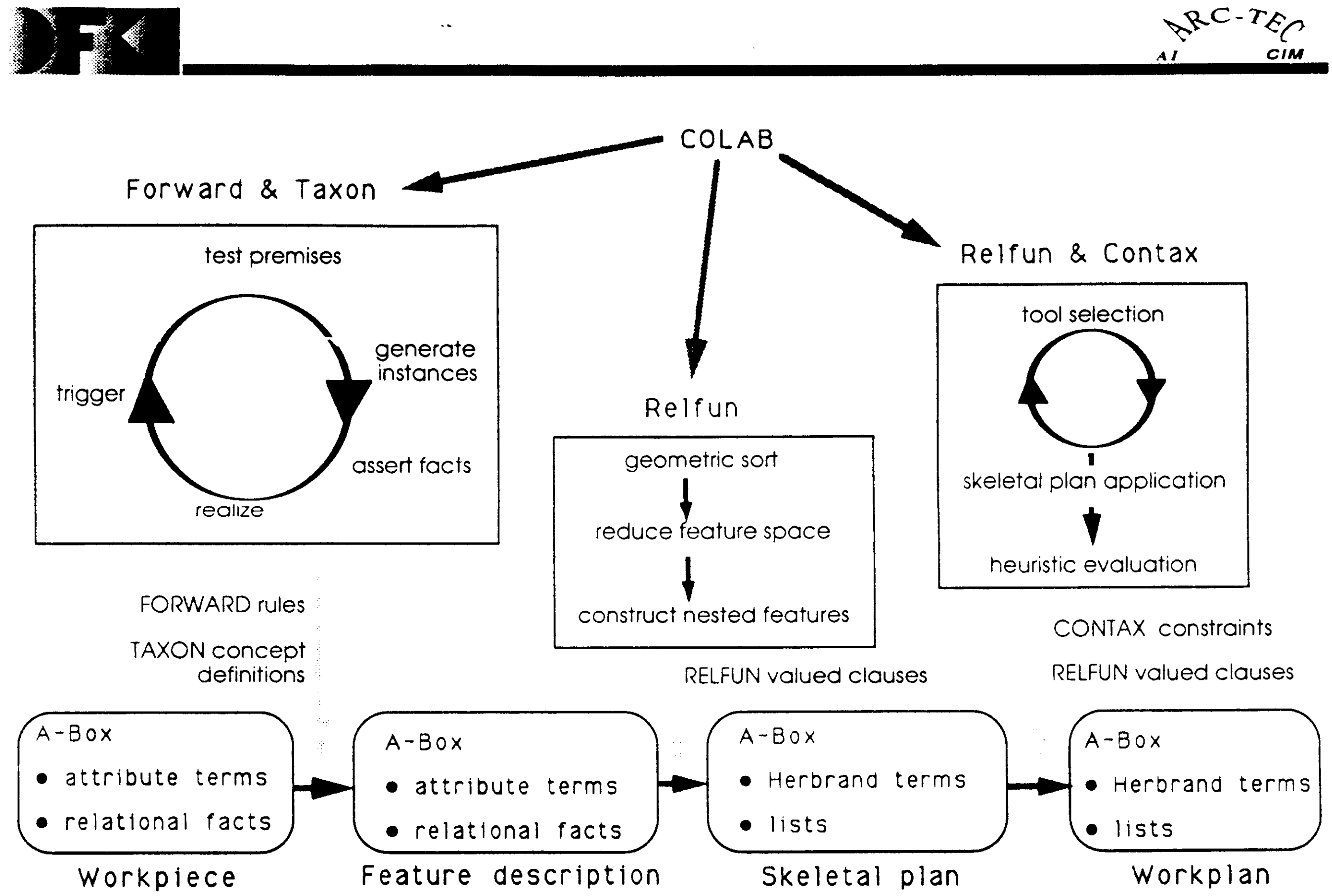


```
;; process-material-tool specifies the usability of cutting-plates w.r.t. the

;; the constraint plate-eangle maps the cutting-plates to their edge-angles


(pc plate-eangle (to pl) (plate-geometries edge-angles)
          (R 0)
          (S 90)
          (T 60)
          (D 55)
;;          (V 35)
          (C 80)
          (K 55))

;; the constraint process-eangle gives expression to the fact that roughing
;; should be performed using a big edge angle whereas for finishing, a small
;; edge angle is appropriate. Round plates (with edge-angle 0) can be used
;; both for roughing and finishing.


(pc process-eangle (pr ea) (processes edge-angles)
          (roughing small-edge-angles)
          (roughing medium-edge-angles)
          (finishing medium-edge-angles)
          (finishing big-edge-angles))

;; The constraint tc-ea-al gives expression to the requirement that
;; the sum of the tool-cutting-edge angle, the tool-edge-angle and the
;; angle alpha must be le 180 degrees.

( lc tc-ea-al (tc ea al) (tc-edge-angles edge-angles acute-angles) le180)

(lc tc-beta (tc bt) (tc-edge-angles acute-angles) ge-or-zero)
```

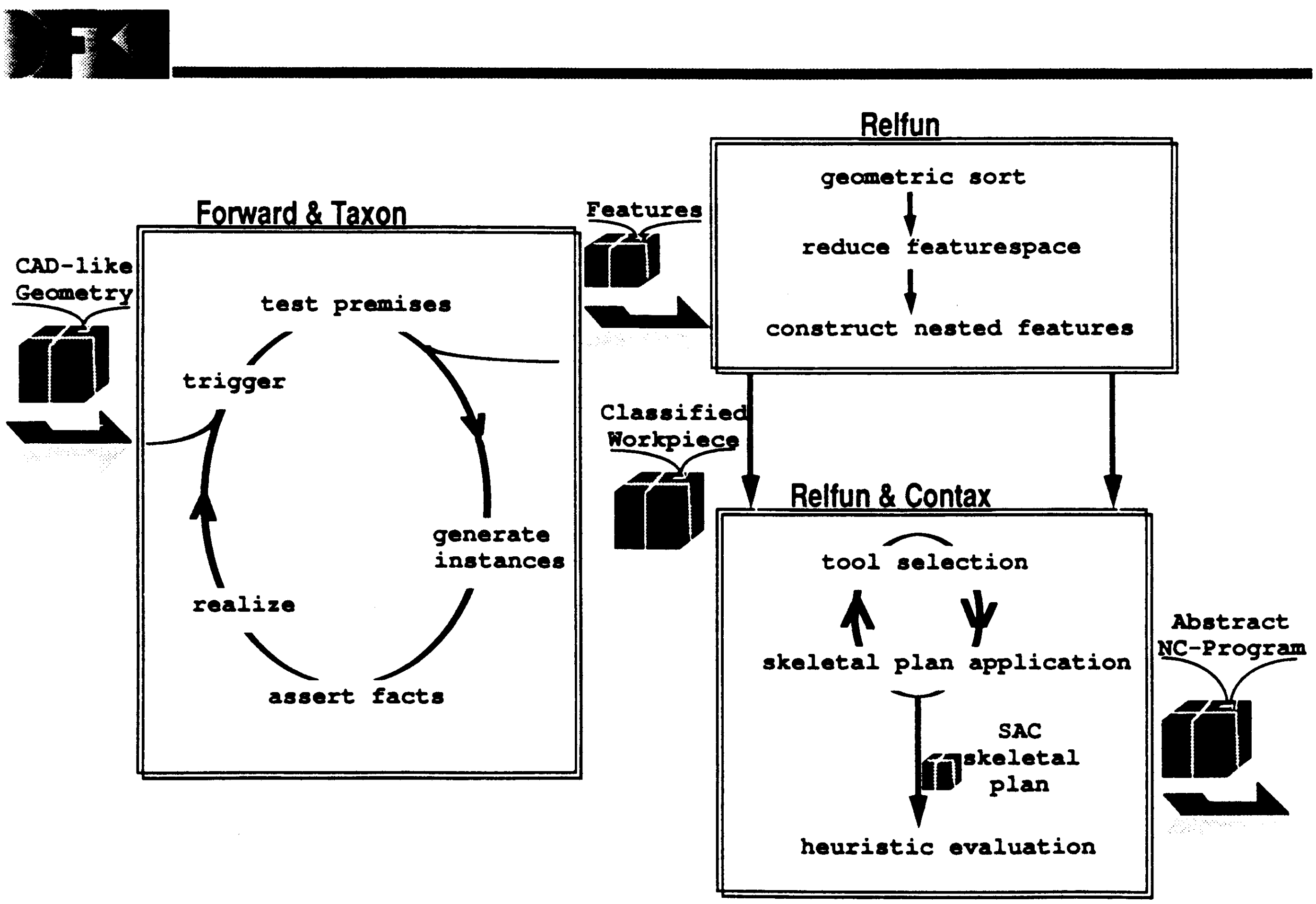
D.2 CONTAX Lisp Functions

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;            microCAD2NC             ;;;
;;            CONTAX part             ;;;
;;        Contax Lisp Functions       ;;;
;; (c) Joerg Mueller  September 1991 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



; first-char(atom char) delivers as result true if the first char of atom
; it is used in order to find the geometry of a cutting plate.

defun first-char (atom1 res)
      (cond
           ((null atom1) nil)
           ( t (eq res (intern (substring (string atom1) 0 1))))))

; le180(W1 W2 W3) is true if the sum of W1, W2 and W3 is less or equal 180

defun le180 (W1 W2 W3)
      (<= (+ (+ W1 W2) W3) 180))


; ge-plus-fa(W1 W2) is true if W1 >= W2 + 5

defun ge-plus-5 (W1 W2)
      (>= W1 (+ W2 5)))

defun ge-or-zero (W1 W2)
      (or
          (>= W1 W2)
          (equal W1 0)
      ))
```

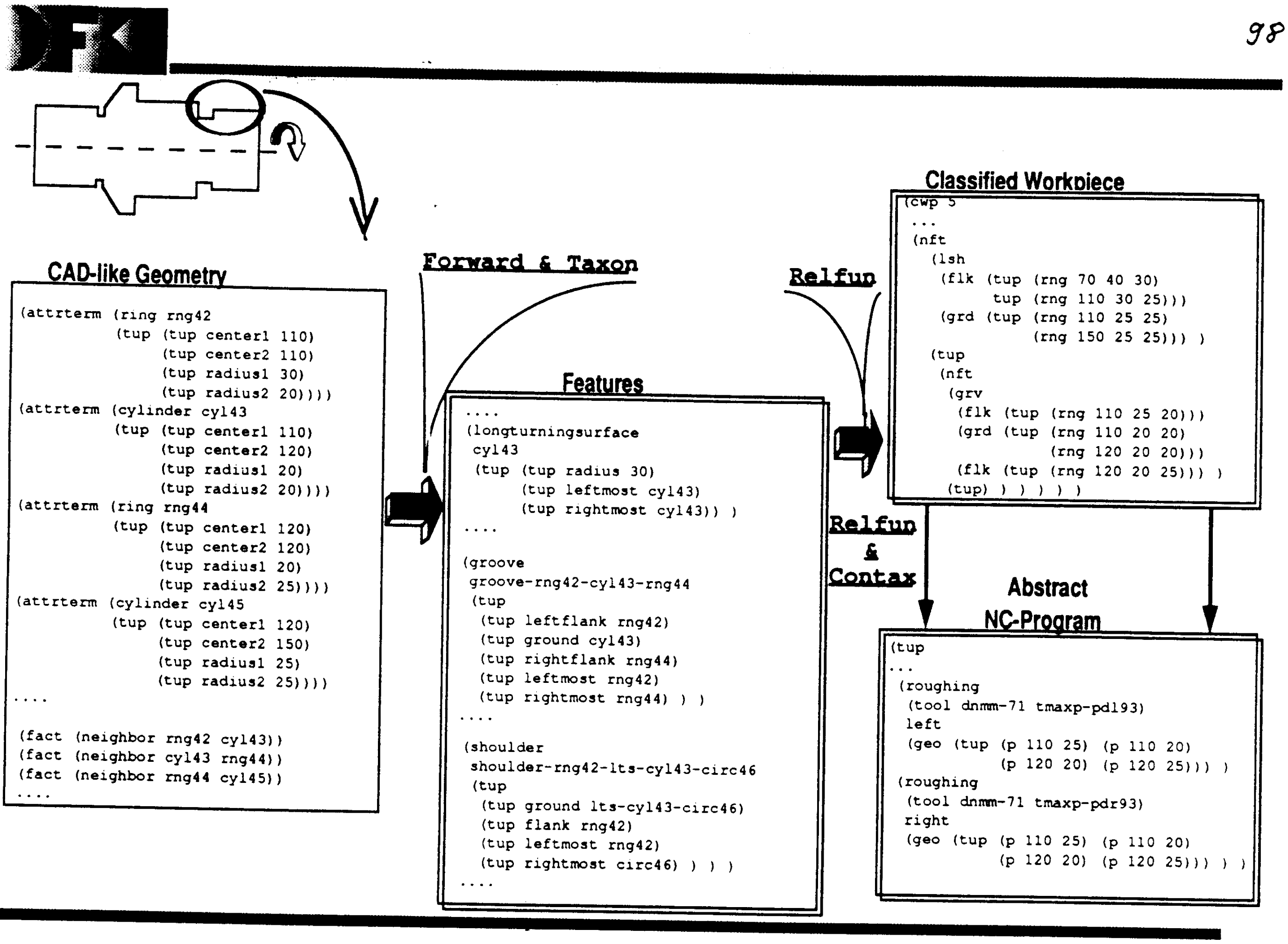
E Figures



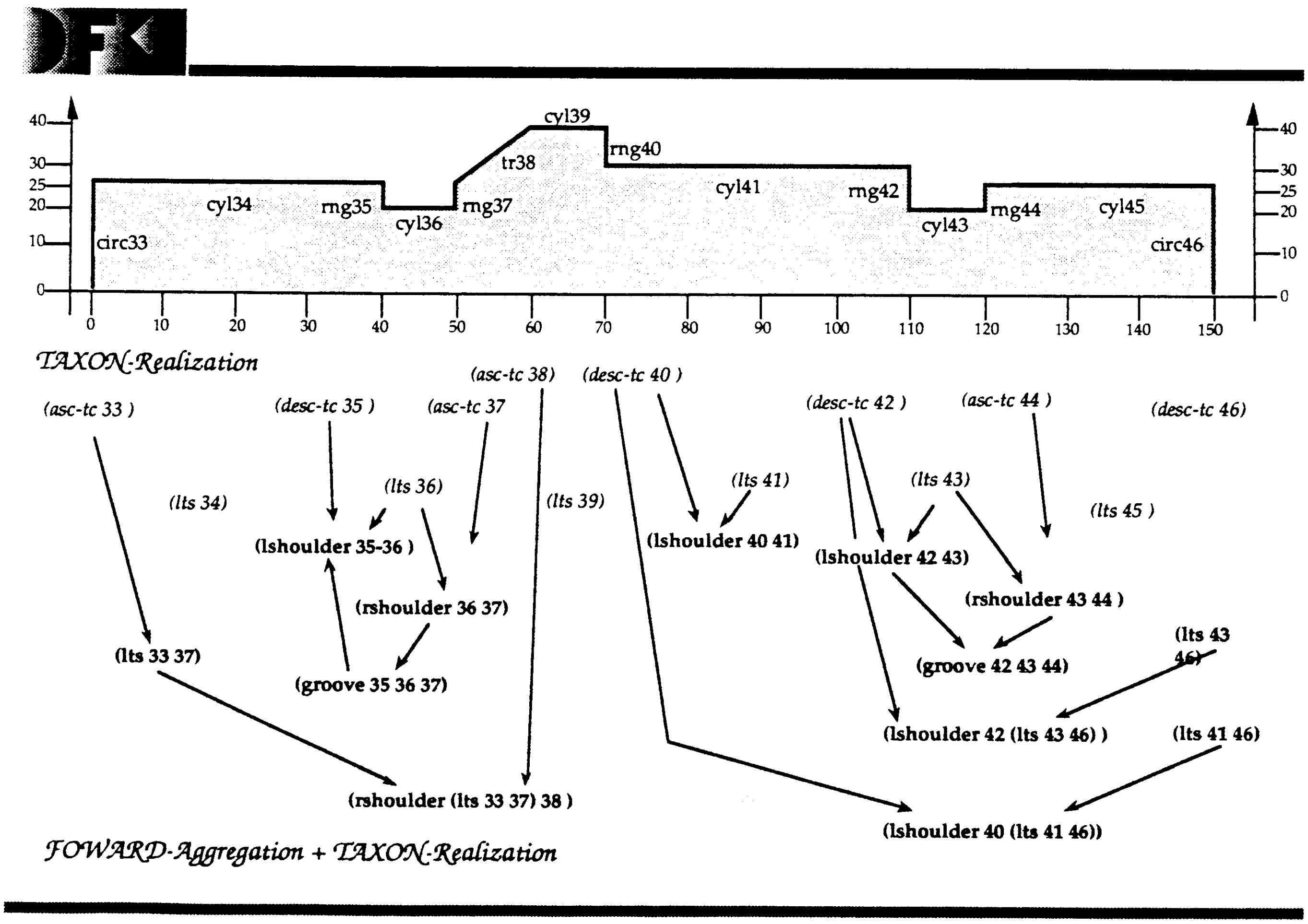
µCAD2NC: Where and How COLAB is used in a Workplanning Model



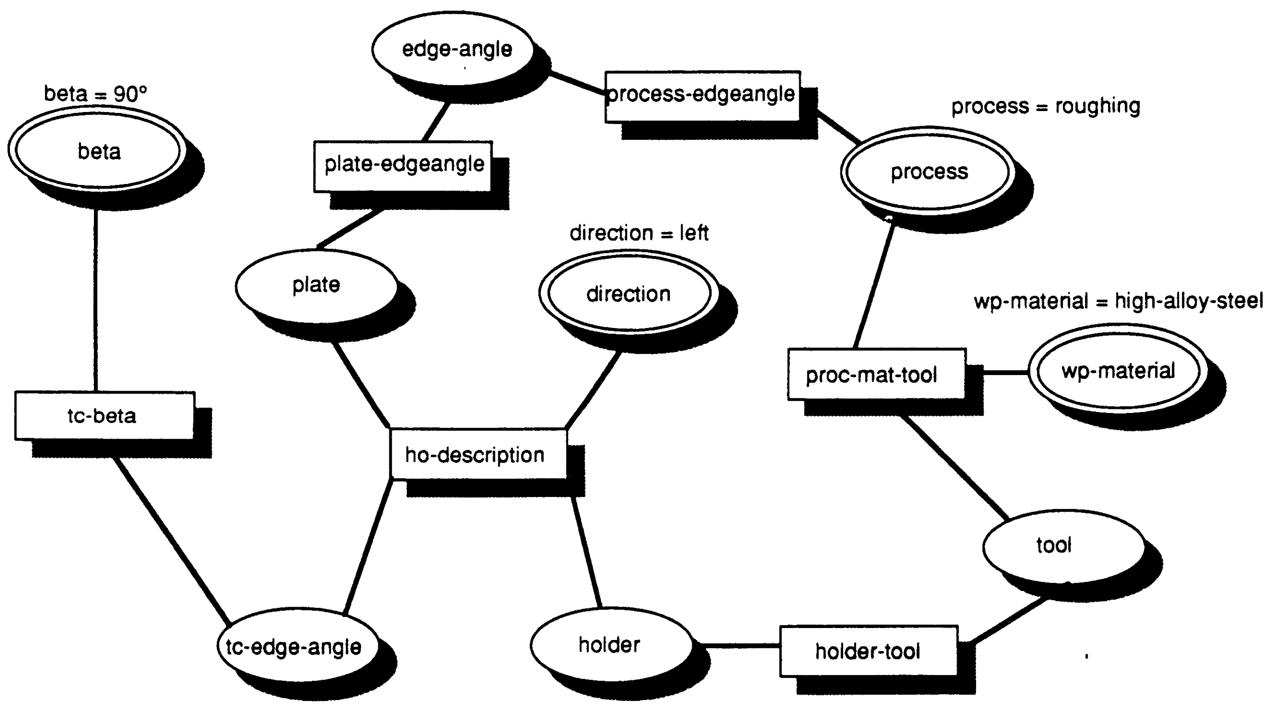
COLAB Subsystems cooperating in µCAD2NC



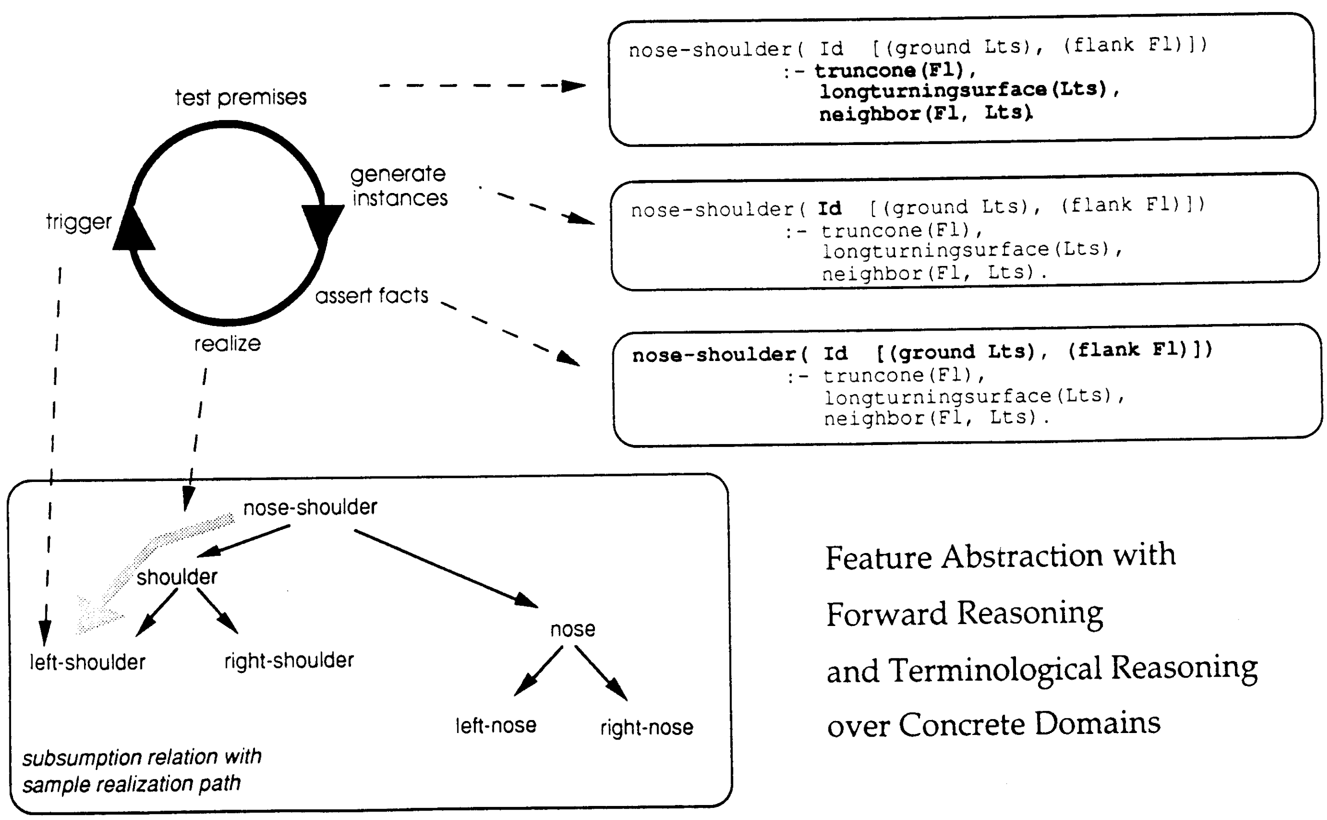
µCAD2NC Data Flow



Feature Recognition in TAXON and FORWARD

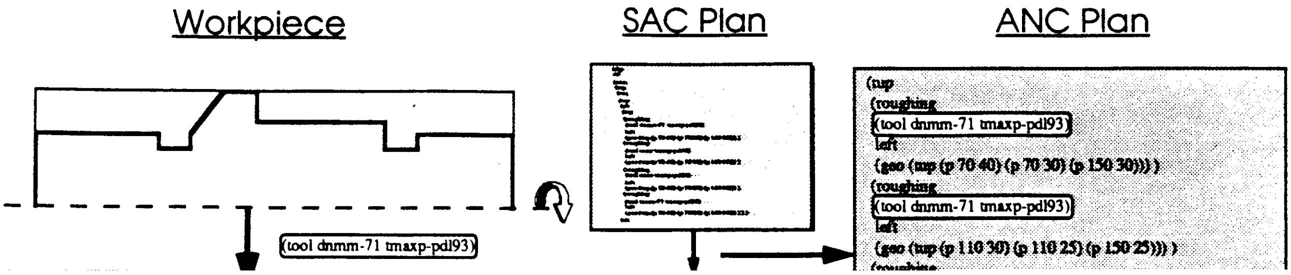


Constraint Net used for tool selection in µCAD2NC



Feature Abstraction with
Forward Reasoning
and Terminological Reasoning
over Concrete Domains

A Tight Coupling of FORWARD and TAXON


Workpiece
SAC Plan
ANC Plan
(tool dnmm-71 tmaxp-pdl93)
(tup
(roughing
(tool dnmm-71 tmaxp-pdl93)
left
(geo (tup (p 70 40) (p 70 30) (p 150 30))))
(roughing
(tool dnmm-71 tmaxp-pdl93)
left
(geo (tup (p 110 30) (p 110 25) (p 150 25))))

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
6750 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all publisched papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

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 Seiten

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-90-17
Stephan Busemann: Generalisierte Phasenstrukturgrammatiken und ihre Verwendung zur maschinellen Sprachverarbeitung
114 Seiten

RR-91-01
Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, 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-91-10
Franz Baader, Philipp Hanschke: A Scheme for Integrating Concrete Domains into Concept Languages
31 pages

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

RR-91-12
J.Mark Gawron, John Nerbonne, Stanley Peters: The Absorption Principle and E-Type Anaphora
33 pages

RR-91-13
Gert Smolka: Residuation and Guarded Rules for Constraint Logic Programming
17 pages

RR-91-14
Peter Breuer, Jürgen Müller: A Two Level Representation for Spatial Relations, Part I
27 pages

RR-91-15
Bernhard Nebel, Gert Smolka: Attributive Description Formalisms ... and the Rest of the World
20 pages

RR-91-16
Stephan Busemann: Using Pattern-Action Rules for the Generation of GPSG Structures from Separate Semantic Representations
18 pages

RR-91-17
Andreas Dengel, Nelson M. Mattos: The Use of Abstraction Concepts for Representing and Structuring Documents
17 pages

RR-91-18
John Nerbonne, Klaus Netter, Abdel Kader Diagne, Ludwig Dickmann, Judith Klein: A Diagnostic Tool for German Syntax
20 pages

RR-91-19
Munindar P. Singh: On the Commitments and Precommitments of Limited Agents
15 pages

RR-91-20
Christoph Klauck, Ansgar Bernardi, Ralf Legleitner FEAT-Rep: Representing Features in CAD/CAM
48 pages

RR-91-21
Klaus Netter: Clause Union and Verb Raising Phenomena in German
38 pages

RR-91-22
Andreas Dengel: Self-Adapting Structuring and Representation of Space
27 pages

RR-91-23
Michael Richter, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Akquisition und Repräsentation von technischem Wissen für Planungsaufgaben im Bereich der Fertigungstechnik
24 Seiten

RR-91-24
Jochen Heinsohn: A Hybrid Approach for Modeling Uncertainty in Terminological Logics
22 pages

RR-91-25
Karin Harbusch, Wolfgang Finkler, Anne Schauder: Incremental Syntax Generation with Tree Adjoining Grammars
16 pages

RR-91-26
M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, G. Merziger:
Integrated Plan Generation and Recognition

RR-91-32
Rolf Backofen, Lutz Euler, Günther Görz:
Towards the Integration of Functions, Relations and Types in an AI Programming Language
14 pages

RR-91-33
Franz Baader , Klaus Schulz:
Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures
33 pages

RR-91-35
Winfried Graf, Wolfgang Maaß: Constraint-basierte Verarbeitung graphischen Wissens
14 Seiten

DFKI Technical Memos

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

TM-91-10
Béla Buschauer, Peter Poller, Anne Schauder, Karin Harbusch: Tree Adjoining Grammars mit Unifikation
149 pages

TM-91-11
Peter Wazinski: Generating Spatial Descriptions for Cross-modal References
21 pages

TM-91-12
Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13
Knut Hinkelmann:
Forward Logic Evaluation: Developing a Compiler

16 pages

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel: ODA-based modeling for document analysis
14 pages

DFKI Documents

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

D-91-02
Jörg P. Müller: Design and Implementation of a Finite Domain Constraint Logic Programming System based on PROLOG with Coroutining
127 pages

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

D-91-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1990
93 Seiten

D-91-06

D-91-07
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner
TEC-REP: Repräsentation von Geometrie- und Technologieinformationen
70 Seiten

D-91-08
Thomas Krause: Globale Datenflußanalyse und horizontale Compilation der relational-funktionalen Sprache RELFUN
137 pages

D-91-09
David Powers and Lary Reeker (Eds):
Proceedings MLNLO'91 - Machine Learning of Natural Language and Ontology
211 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

Donald R. Steiner, Jürgen Müller (Eds.):
MAAMAW'91: Pre-Proceedings of the 3rd European Workshop on „Modeling Autonomous Agents and Multi-Agent Worlds"
246 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-11
Thilo C. Horstmann: Distributed Truth Maintenance
61 pages

D-91-12
Bernd Bachmann:
$H^{iera}C_{on}$ - a Knowledge Representation System with Typed Hierarchies and Constraints
75 pages

D-91-13
International Workshop on Terminological Logics
Organizers: Bernhard Nebel, Christof Peltason, Kai von Luck
131 pages

D-91-14
Erich Achilles, Bernhard Hollunder, Armin Laux, Jörg-Peter Mohren: *KRIS*: Knowledge Representation and Inference System
- Benutzerhandbuch -
28 Seiten

D-91-15

D-91-15
Document

μCAD2NC: A Declarative Lathe-Workplanning Model Transforming CAD-like Geometries into Abstract NC Programs

Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch,
Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, Frank Steinle