Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-92-35

Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment

Manfred Meyer

August 1992

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, 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

Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment

Manfred Meyer

DFKI-RR-92-35

This paper will also be published by AAAI Press in the Proceedings of the *Fifth International Symposium on Artificial Intelligence ISAI'92 — The Artificial Intelligence Technology Transfer Conference*, Cancun, Mexico, December 7–11, 1992.

This work has been supported by The Federal Ministry for Research and Technology (BMFT) under grant ITW 8902 C4.

Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment

Manfred Meyer
German Research Center for Artificial Intelligence (DFKI)
P. O. Box 20 80, D-6750 Kaiserslautern, Germany
E-mail: meyer@dfki.uni-kl.de

Abstract

In this paper we shall discuss how to treat the automatic selection of appropriate lathe tools in a computer-aided production planning (CAPP) application as a constraint satisfaction problem (CSP) over hierarchically structured finite domains. Conceptually it is straightforward to formulate lathe-tool selection in terms of a CSP, however the choice of constraint and domain representations and of the order in which the constraints are applied is nontrivial if a computationally tractable system design is to be achieved. Since the domains appearing in technical applications often can be modeled as a hierarchy, we investigate how constraint satisfaction algorithms can make use of this hierarchical structure. Moreover, many real-life problems are formulated in a way that no optimal solution can be found which satisfies all the given constraints. Therefore, in order to bring AI technology into real-world applications, it becomes very important to be able to cope with conflicting constraints and to relax the given CSP until a (suboptimal) solution can be found. For these reasons, the constraint system CONTAX has been developed, which incorporates an extended hierarchical arc-consistency algorithm together with discrete constraint relaxation and has been used to implement the lathe-tool selection module of the ARC-TEC planning system.

Area of application: computer-integrated manufacturing, computer-aided production planning

AI techniques: hierarchical constraint satisfaction, constraint relaxation

1 Introduction

The rapid development of manufacturing and computer technologies has generated new problems. To solve these problems modern tools and techniques are required. Artificial Intelligence (AI) is one of the most appropriate techniques for solving complex industrial problems [Kusiak, 1988].

The ARC-TEC project at DFKI constitutes an AI approach to implement the idea of computer-integrated manufacturing (CIM). Along with conceptual solutions, it provides a continuous sequence of software tools for the **A**cquisition, **R**epresentation, and **C**ompilation of **TEC**hnical knowledge (cf. [Bernardi *et al.*, 1991]). This shell combines the KADS knowledge-acquisition methodology, the KL-ONE representation theory [Brachman and Schmolze, 1985], and WAM compilation [Hein and Meyer, 1992] and constraint-handling technologies [Meyer *et al.*, 1992]. For its evaluation, an expert system for production planning has been developed.

The input to the production planning system is a very low-level description of a rotational-symmetric workpiece as it comes from a CAD system. Geometrical description 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 [Schmalhofer *et al.*, 1991] have a library of skeletal plans in their minds. Each of these plans is associated with a more or less abstract description of a (part of a) workpiece, which are called workpiece features [Klauck *et al.*, 1991]. Such a feature is defined by its association to a corresponding manufacturing method. The generation of an abstract feature description of the workpiece is the first step of the production planning process. The obtained features characterize the workpiece with respect to its production. In a second step the skeletal plans (associated to the features) are retrieved and merged resulting in an abstract NC program, which is then transformed into code for the concrete CNC machine.

The planning system has been developed using CoLab [Boley *et al.*, 1991], a hybrid-knowledge compilation laboratory which integrates the power of forward and backward reasoning, constraint propagation, and taxonomic classification. The focus is not on an integrated smooth system, but on exemplifying methodologies for the use of hybrid formalisms at certain subtasks. These various subtasks require a number of specialized reasoning mechanisms integrated in CoLab: Feature aggregation is performed by the forward reasoning component, Forward[1],

[1] Forward [Hinkelmann, 1992] 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 different implementations: 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 code for an extended WAM with a special forward code area.

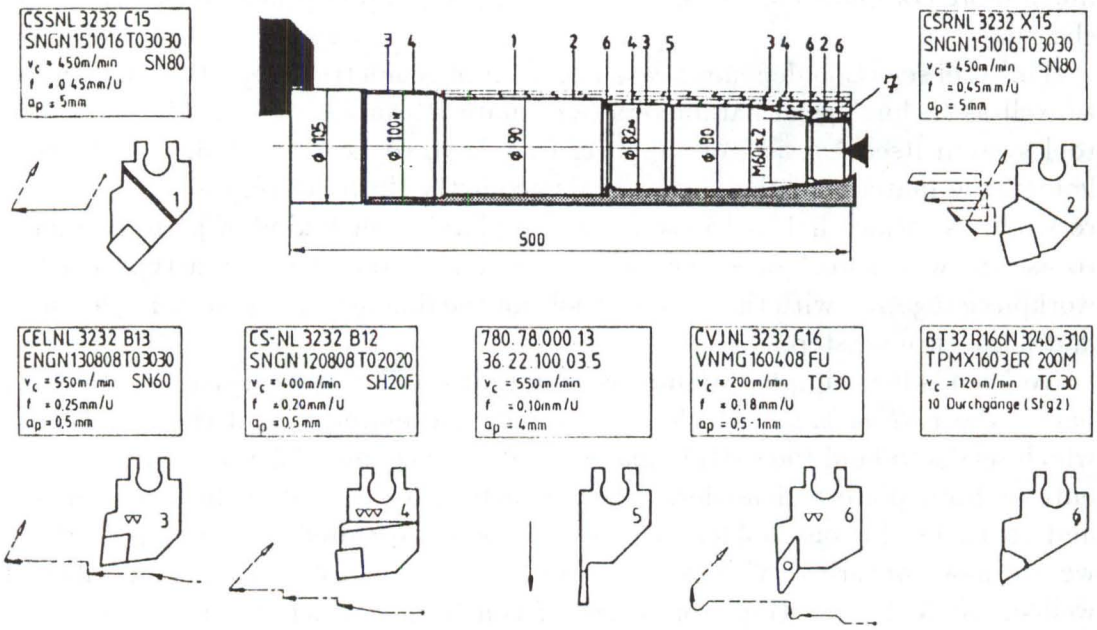

Figure 1: An example workpiece with its selected lathe tools

together with the terminological component, TAXON[2]. The derived features are finally collected by a program written in RELFUN [Boley, 1990], the backward reasoning component of COLAB. The abstract NC program is then generated from the classified workpiece by parameterized retrieval of skeletal plans using RELFUN and selecting the appropriate lathe tools via CONTAX, the constraint propagation component of COLAB.

2 The lathe-tool selection problem

The application problem we are dealing with for the rest of this paper will be to find appropriate lathe tools to manufacture the workpiece. According to the shape, the material and other attributes of the lathe part to be manufactured, the work-plan consists of a number of different steps. A typical work-plan may provide one step for roughing, another step for finishing and a third (facultative) step for doing the fine finishing of the lathe part. However, a work-plan can be

[2]TAXON [Baader and Hanschke, 1991] 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 the concepts can be instantiated by individuals. The individuals have attributes and belong to concepts. This assertional knowledge is used to determine the most specific concepts in the subsumption hierarchy to which the individuals belong (*realization service*).

much more complicated. For each processing step, appropriate tools have to be chosen.

This tool selection depends heavily on a lot of geometrical (e.g. the edge-angle) as well as technological parameters (e.g. material, process etc.). Moreover, the tool system itself consists of subparts that have to be combined, e.g. the tool holder, the material of the plate and its geometry. In practice, there are a lot of restrictions, 'which holder to use for which plate', 'which kind of plate geometry to use for which workpiece' contour and so on. Figure 1 shows a typical lathe workpiece together with the selected tools for the different manufacturing features and lathe-turning steps.

To keep things simple, we may assume that a lathe tool consists of two basic parts: the *cutting plate*, which actually cuts the material, and the *tool holder*, which serves to hold the cutting plates. We can exchange either the cutting plate only or both plate and holder. There is a functional relation between holders and tools, i.e. for one holder, only a few tools are suited. In our application, we are now concerned with finding a well-suited tool—or rather: a number of well-suited tools—starting from a set of constraints which describe the actual problem, i.e. information about the process to be performed, about the lathe part to be processed, and internal information about the compatibility of holders and cutting-plates as well as about holder and plate geometries. Lathe-tool selection will then result in a set of possible holder/tool combinations for each skeletal plan or manufacturing feature. Using this information, the planning layer formalized in RELFUN will finally perform the optimizations necessary to obtain a (sub)optimal work-plan.

3 Formalizing the lathe-tool selection problem as CSP

When formalizing the tool selection problem as a CSP, the first thing we have to do is to restrict the number of input parameters, which crucially determines the complexity of the problem, since each parameter corresponds to a variable in the constraint net. For our small example we will use the following variables:

- **Holder**: This variable denotes the tool holder. In the beginning, it ranges over the domain of all holders. During constraint propagation, it will be restricted to the set of holders which can currently be chosen.
- **Plate**: This variable denotes the cutting plate to be chosen. Analogously to the holder variable, it ranges over the set of all cutting plates and will be restricted subsequently.
- **Process**: This variable corresponds to the actual kind of processing.
- **WP-material**: This variable contains the material of the lathe workpiece.

- **Beta-max**: This variable denotes the maximal angle β appearing within the range of one feature of the workpiece.[3]

- **Edge-Angle**: This variable embodies the most important geometrical attribute of a cutting-plate, its edge-angle ε.

- **TC-Edge-Angle**: The tool cutting edge-angle χ is a geometrical characteristic of the tool holder. It denotes the angle between the horizontal cutting direction and the marginal cutting axis of the holder.

Figure 2 gives a better understanding of the geometrical items introduced above.

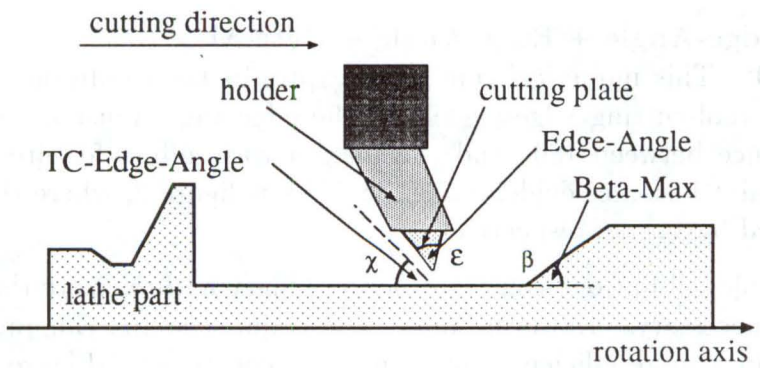

Figure 2: The Angle Constraint

Having identified the problem variables, the constraints can be put on the variables. In the following, we will consider only the most important constraints:

- **holder_tcea(Holder, TC-Edge-Angle)**: This constraint describes the functional relation between a holder and its tool-cutting edge-angle. It is represented as a primitive or database constraint by enumerating all the possible combinations.

- **plate_ea(Plate, Edge-Angle)**: This constraint is a database-constraint, too. It denotes the fact that each plate has its own edge-angle[4].

- **compatible(Holder, Plate)**: This constraint expresses the compatibility condition between tool holders and cutting plates.

[3] In general, each of these features corresponds to a single working process.

[4] Of course, we could have implemented the plate as a more complex data structure containing its edge-angle as an attribute. For the sake of uniformity, we implemented it as a constraint, just as we did with the **holder_tcea** constraint.

- **hard_enough(Plate, WP-Material)**: For materials with different degrees of hardness, different cutting-plates have to be used. Processing hardened steel, e.g., may require ceramic or even diamond cutting plates, whereas aluminum can be cut with other, cheaper plates. Note, however, that hardness is just one of many attributes of a material which are important in order to choose the right cutting plate.

- **process_holder(Process, Holder)**: For the different steps of processing, different types of holders are appropriate. Well-suited for the purpose of roughing, e.g., are holders of the CSSNL class.

- **process_edge_angle(Process, Edge-Angle)**: This constraint expresses a rule of thumb which says that for roughing, plates with big edge-angles should be chosen, whereas for finishing, smaller edge-angles are appropriate.

- **TC-Edge-Angle + Edge-Angle + Beta-Max $<$ 180°**: This numerical constraint expresses the condition that the sum of the tool-cutting edge-angle and the edge-angle must be less than the difference between 180° and the maximal ascending feature-angle. This constraint becomes evident when looking at figure 2, where the angles are denoted by χ, ε, β, respectively.

In our example, all but one constraint are of binary nature. Note that in general this is not necessary. However, many constraint systems can process binary constraints in a more efficient way than n-ary constraints. Figure 3 shows the resulting constraint net for our application example.

4 The constraint system CONTAX

Various approaches and algorithms have been developed to tackle the constraint satisfaction problem (CSP). The computational complexities of these algorithms heavily depend on the level of consistency they compute (cf. [Mackworth and Freuder, 1985]). To reduce the complexity, terminological knowledge can be used to structure the domains of the variables occurring in the CSP.

The constraint system CONTAX supports constraint propagation methods for computing locally or globally consistent assignments of values from the given domains to the variables of the CSP. Especially, CONTAX provides a mechanism for solving constraints over hierarchically structured domains which can be defined using the terminological language TAXON.

4.1 Constraint satisfaction and local consistency

Given a set of n variables, each with an associated domain and a set of constraining relations each involving a subset of the variables, a constraint satisfaction problem can informally be defined as to find all possible n-tuples such that

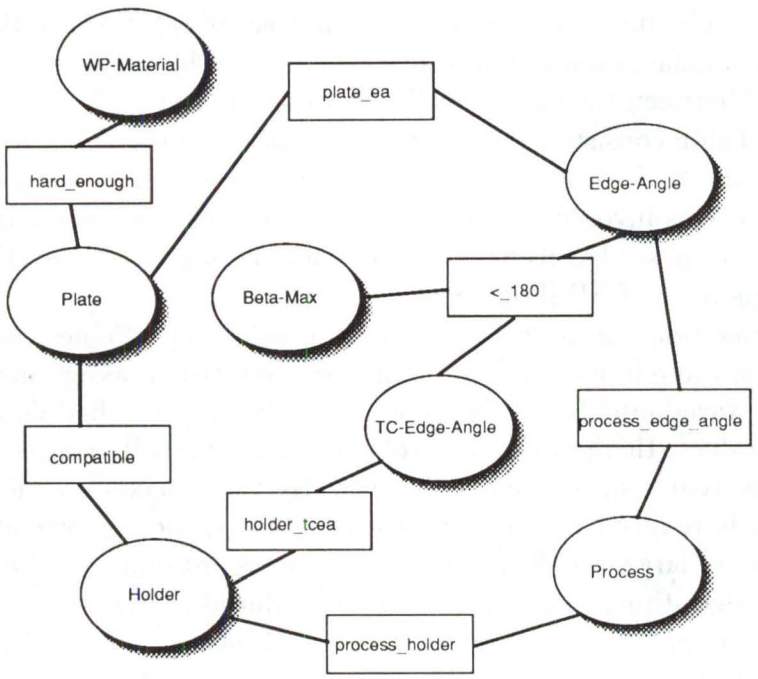

Figure 3: The Exemplary Constraint Net

each n-tuple is an instantiation of the n variables satisfying the relations. The constraining relations are called *constraints*. Constraints may be of any arity, whereas many constraint systems restrict them to be unary or binary. The variables of the CSP together with the constraints defined over them can be regarded as a constraint graph or *constraint net*. More formally, the general constraint satisfaction problem can be defined as follows:

Definition 1 (CSP)
Assume a finite set of variables $V = \{X_1, \ldots, X_n\}$, *a finite set* $D = D_1 \cup \ldots \cup D_n$ *(domains), and a finite set* R *of relations* R_i *are given, such that* $R_i \subseteq D_{i_1} \times \ldots \times D_{i_{k_i}}$, *where* $D_i \subseteq D$ *and* k_i *is the arity of* R_i. *The constraint satisfaction problem is to find an assignment* $\sigma : V \rightarrow D$ *for the variables such that all constraints are satisfied simultaneously.*

A common example of a constraint satisfaction problem is the graph-coloring problem. Since graph-coloring is an NP-complete CSP, it is most unlikely that a polynomial time algorithm exists for solving general CSPs. However, a number of algorithms based on local propagation have been developed. These algorithms do not necessarily solve a CSP completely but they eliminate, once and for all, local inconsistencies that cannot participate in any global solution. These inconsistencies would otherwise have been repeatedly discovered by any backtracking solution. Hence local consistency algorithms can play the role of a preprocessor

for subsequent backtracking search, or they can be coupled with case analysis or simple domain splitting to recover the complete set of solutions to the CSP.

Constraint satisfaction algorithms can be classified by the level of consistency they establish between the variables of the constraint net. A k-consistency algorithm removes all inconsistencies involving all subsets of size k of the n variables. For example, the *node*, *arc*, and *path* consistency algorithms detect and eliminate inconsistencies involving $k = 1$, 2, and 3 variables, respectively. Freuder's generalization of those algorithms for $k \rightarrow n$ can be used to produce the complete set of solutions to the CSP [Freuder, 1978].

Local propagation computes arc- or path-consistency. Values not appearing in any solution are eliminated. Local consistency gives an assignment of sets of values to the variables. Since the constraints are not satisfied simultaneously by the same values, this relaxes the problem. Every globally consistent solution is locally consistent too, but not vice-versa. By that relaxation the complexity of algorithms is reduced to polynomial time. Thus, local propagation can be used efficiently in large search spaces to pre-process and improve the behavior of backtracking algorithms, which run over the reduced search space afterwards.[5] Nevertheless, the pruning effect by local propagation depends on the kind of the problem: local propagation does not necessarily reduce the search space at all. On the other hand, some special instances of the CSP can be solved without any backtracking, provided there is some additional information about the structure of the constraint net [Meseguer, 1989].

4.2 Hierarchically structured domains and hierarchical arc-consistency

In many real-world applications, objects can be clustered and grouped to classes according to some of their properties. These classes often form a hierarchy, which can be described by a subclass-superclass relation, also called `isa`-relation. Knowledge representation using `isa`-hierarchies enables efficient use of attributes and properties of the considered domains. The transitivity of the `isa`-relation guarantees the inheritance of properties from super- to subclasses; subclasses can be seen as specializations. Any arbitrary domain can be transformed into a directed acyclic graph (*DAG*), which describes the domain as a hierarchy—in the worst case only consisting of nodes. In principle, the algorithms for solving any CSP, especially those dealing with large domains and hence large search spaces, can make use of structuring the domains.

The class hierarchy can be defined directly by enumerating the `isa`-links between classes. Moreover, the classes may declaratively be defined in terms of concept definitions in the sense of terminological languages like KL-ONE, which

[5] The algorithm AC-3 [Mackworth, 1977] for achieving arc-consistency has a time complexity of $O(ea^3)$. Its space complexity is $O(e + na)$, with e constraints (arcs), n variables (nodes) and a cardinality of a for all domains.

then are classified to get the subsumption hierarchy. For this purpose, CONTAX employs the terminological language TAXON and its classification algorithms to structure the domains. The classifier does usually not build a tree, since one concept may be a subconcept of more than one other. More likely, the resulting hierarchy describing the domain of some constraint variables becomes a DAG representing this lattice.

To exploit the hierarchical structure of domains, the propagation algorithms had to be extended to deal with concepts instead of elements of a domain. The main aim is to reduce the complexity measured by the number of evaluations of the constraining relations. Therefore, CONTAX provides an extended and improved version of the hierarchical arc-consistency algorithm (HAC) presented in [Mackworth *et al.*, 1985]. It uses two new predicates, which evaluate the constraints between arbitrary concepts by using inheritance mechanisms. Since the concepts represent a large number of elements at once, this improves the efficiency of the propagation algorithm.

In [Mackworth *et al.*, 1985] some assumptions about the constraints and hierarchies are made. The HAC algorithm only deals with binary constraints over binary, singly rooted, strict trees as domains. For any real-world CSP the restrictions made by HAC seem to be inappropriate. In addition to that, it is useful—especially for large domains—to allow definitions of constraints between arbitrary concepts. Since the hierarchies are seen as structured inheritance networks, we have to make clear what inheritance means for constraints:

Definition 2 (inheritance of constraints)
Let $R_j \subseteq D_1 \times \ldots \times D_i \times \ldots \times D_k$ *be a constraint,* $(d_1, \ldots, d_i, \ldots, d_k) \in R_j$ *be a tuple in* R_j. *Then for all* $\tilde{d}_i \in D_i$ *the tuple* $(d_1, \ldots, \tilde{d}_i, \ldots, d_k) \in R_j$ *iff* $\texttt{isa}(\tilde{d}_i, d_i)$.

A locally consistent value assignment can be defined in terms of hierarchical arc-consistency. For simplicity reasons, we only give the definition for binary constraints here.[6] However, the actual CONTAX implementation uses an extended notion dealing with n-ary constraints:

Definition 3 (arc-consistency)
A value assignment $\sigma : V \rightarrow 2^D$ *of a set of values to each variable of the constraint net with* $\sigma(X_i) = \tilde{D}_i \subseteq D_i$ *is* **arc-consistent** *iff for all variable pairs* $(\tilde{X}_i, \tilde{X}_j)$ *and for all constraints* R_{ij} *defined over them it holds that for each* $d \in \tilde{D}_i$ *there exists at least one* $\tilde{d} \in \tilde{D}_j$ *such that the pair* $(d, \tilde{d})$ *satisfies the constraint* $R_{i,j}$, *that is* $R_{ij}(d, \tilde{d})$ *holds.*

The image $\tilde{D}$ of the value assignment σ only includes the most universal concepts that establish arc-consistency.

Hierarchical arc-consistency can now be defined based on the inheritance of constraints through `isa`-links:

[6] Since any n-ary relation can be expressed as a set of binary relations, this is no serious restriction in theory.

Definition 4 (hierarchical arc-consistency)
A value assignment $\sigma : V \rightarrow 2^D$ *with* $\sigma(X_i) = \tilde{D}_i \subseteq D_i$ *is* **hierarchically arc-consistent**, *if it is arc-consistent and most universal, i.e. for all* $d \in \tilde{D}_i$ *there does not exist a more general concept* $q \in D_i, q \neq d$ *and* `isa`(d, q) *such that the assignment*

$$\tilde{\sigma}(X_k) = \begin{cases} (\tilde{D}_k \setminus d) \cup \{q\} & if \quad k = i \\ \sigma(X_k) & if \quad k \neq i \end{cases}$$

is arc-consistent, too.

The hierarchical constraint satisfaction problem (HCSP) is then to compute a hierarchical arc-consistent value assignment which can then using backtracking be further restricted towards a globally consistent value assignment satisfying all constraints simultaneously.

4.3 The User's View on CONTAX

Using CONTAX to formalize and solve a constraint satisfaction problem in principle involves the following steps:

- identifying the variables and constraints that constitute the given problem and defining the domains over which the variables range,
- defining the problem constraints,
- connecting variables and constraints to build the constraint net, and
- finally propagating some initial value assignments through the constraint net to restrict the domains of the variables and to achieve a solution for the underlying HCSP.

4.3.1 Defining domains

In its simplest form, plain domains can be defined by simply enumerating all the elements belonging to the domain. For example, the statement

```
(def-domain alloy-steel
  (low-alloy-steel high-alloy-steel))
```

introduces a new domain of some workpiece materials. Using the hierarchical structure of the domain, a statement like

```
(def-domain steel
  (building-steel alloy-steel
   stainless-steel))
```

defines the hierarchical domain `steel` to be the union of some more specialized domains which have been defined before as plain or even hierarchical domains.

If some considered domain relates to a terminology defined using the KL-ONE-like terminological reasoning system TAXON, the terminology along with all its concepts can be imported and used by CONTAX, e.g. via a statement

```
(import-terminology workpiece-materials).
```

The classified concepts (i.e., the subsumption dag) can directly be used as the domain hierarchy for CONTAX, where the TAXON Abox individuals serve as leaves.

To be usable in real-life applications, CONTAX also supports variables ranging over intervals of integers. Such interval domains, e.g. the domain of angles defined by

```
(def-intdomain angle (0 90)),
```

can directly be mapped to hierarchical domains.

4.3.2 Defining constraints

CONTAX provides different types of constraints: primitive (or extensional), predicative and compound constraints. All constraint types may be defined over any number of variables.

Primitive constraints are defined by enumerating all the tuples satisfying the constraint. This kind of constraint can also be regarded as a *database constraint*. One step towards a more comfortable definition of constraints is to make use of non-leaf concepts when enumerating the relations. Consider, for example, the following constraint defining compatibility between workpiece material and cutting-plates:

```
(def-primitive-constraint compatible
  :interface (material plate)
  :domains (material plate)
  :tuples
    ((cast cnmm) ...
     (alloy-steel dnmm-41) ...
     (steel dnmm-71) ...))
```

Here the fact that all kinds of `steel` are compatible with the `dnmm-71` plate are expressed by simply including the 'abstract' tuple `(steel dnmm-71)` instead of all the tuples for different kinds of `steel`.

Some constraints occurring in a real-world application are difficult or even impossible to be explicitly enumerated as primitive constraints. This is true, for example, for numerical constraints which should be evaluated by the underlying LISP system. Therefore, constraints can also be defined by providing a LISP function or `lambda`-expression (as argument to the `:predicate` keyword) which then

will be evaluated to test a given tuple for membership in the relation. Consider, for example, the <_180 constraint in the lathe-tool application:[7]

```
(def-lisp-constraint <_180
  :interface
    (beta-max tc-edge-angle edge-angle)
  :predicate
    (lambda (beta-max tcea edge-angle)
      (< (+ beta-max tcea edge-angle)
         180)))
```

Other constraints may be defined by a RELFUN procedure. In this case, for each tuple which has to be tested for membership in the relation, the appropriate RELFUN goal is evaluated.

Often it may happen that the same constraint subnet occurs many times between different variables of the entire CSP. Therefore, it becomes very useful to define this subnet as a compound constraint which itself represents an entire constraint net. *Local variables* of the constraint subnet that only serve to connect local constraints need not to occur in the :interface list. For example, the exemplary constraint net in Figure 3 can be defined as a single compound constraint named tool_sel. The local variable edge_angle is determined by the variable plate and therefore need not to be visible from outside the tool_sel constraint:

```
(def-compound-constraint tool_sel
  :interface
    (holder plate process wp-material
     beta-max tc-edge-angle)
  :constraints
    ((holder_tcea holder tc-edge-angle)
     (plate_ea plate edge-angle)
     (<_180 tc-edge-angle
            edge-angle
            beta-max)
     (compatible holder plate)
     (hard_enough plate wp-material)
     (process_holder process holder)
     (process_edge_angle process
                         edge-angle)))
```

[7]Although the variables beta-max, edge-angle, and tc-edge-angle range over finite discrete domains and it therefore would be possible to explicitly enumerate all tuples satisfying the <_180 constraint, in practice it is much more comfortable and even more efficiently computable to define this constraint as a predicative constraint using the underlying LISP system.

4.3.3 Building the constraint net

Since CONTAX is implemented in an object-oriented fashion based on CLOS, variables and constraints are realized as CLOS objects. Therefore, the constraint net is simply built by creating instances of the variable and constraint objects. Variable instances are created by e.g.

```
(make-variable :name MVAR
               :domain material)
(make-variable :name PVAR
               :domain plate)
```

After having created all needed instances of the problem variables, they can be linked by creating the constraint instances, e.g. an instance of the `compatible` constraint:

```
(make-constraint :name C1
                 :type compatible
                 :material MVAR
                 :plate PVAR
                 :weight hard)
```

4.3.4 Providing information for constraint relaxation

Many real-life problems are *over-specified*, that is, they are formulated in such a way that no single optimal solution satisfying all the given constraints can be found. In order to bring AI technology into real-world applications it becomes very important to be able to cope with conflicting constraints and to provide a mechanism to relax the given CSP until a (suboptimal) solution can be found.

Therefore, CONTAX allows to specify additional information to be used if the constraint problem turns out to be over-specified and the constraint solver detects an inconsistency:

- First, constraints can be weighted. That is, each constraint can be attached to a weight out of the discrete set of five weights ranging from `soft` to `hard`. By default, CONTAX regards all constraints to be `hard` constraints which cannot be relaxed.[8]
- Second, the tuples constituting primitive constraints can be weighted, too. The tuples listed as argument for the `:tuples` keyword in the constraint definition form the minimum definition of the relation. For relaxation purposes the relation can be extended by including additional tuples. These

[8]It has been shown, e.g. by [Descotte and Latombe, 1985], that in general only a small number of discrete weights is necessary to represent the user's knowledge about priorities between constraints. Providing a larger or even an infinite set of weights would violate the declarative style of knowledge processing using constraints, since the user would then be able to directly code control information into the priorities of the constraints.

relaxation tuples are also given a weight ranging over the discrete set from `relax1` to `relax5`. Obviously, this results in a more fine-grain relaxation behavior than weighting entire constraints only.

By this means, CONTAX offers a very declarative and natural way to specify, for example, that the plate `dnmm-41` may also be used for `stainless-steel` if other (better) combinations cannot be used for some reason:

```
(def-primitive-constraint compatible
   :interface (mat pl)
   :domains (material plate)
   :tuples
     ((cast cnmm) ...
      (alloy-steel dnmm-41) ...
      (steel dnmm-71) ... )
   :relax1
      ((stainless-steel dnmm-41)))
```

4.3.5 Computing a hierarchical arc-consistent value assignment

After having defined all variables, constraints, and their connections forming a constraint net, CONTAX is ready to perform its real job, namely to propagate value restrictions through the constraint net in order to compute a hierarchical arc-consistent value assignment.

The basic idea of the CONTAX constraint propagation algorithm is first to start with the mostly relaxed problem, to check whether a hierarchical arc-consistent value assignment can be computed, and then to strengthen the problem step by step until it either becomes equal to the `hard` problem formulation (without using any relaxation tuples) or an inconsistency occurs, in which case the most previously computed hierarchical arc-consistent value assignment is returned:

1. All constraints are relaxed as far as possible, i.e. all relaxation tuples and all `hard` constraints are taken into account. The relaxation value assignment σ_r is set to be the empty assignment $\sigma_\perp : V \to \perp$. All constraints are pushed onto a queue Q of constraints that have to be revised, that is, checked for hierarchical arc-consistency.

2. A constraint $C(X_1, \ldots, X_n) \in Q$ is selected to get revised and is deleted from Q. The domains of the variables $X_1, \ldots, X_n$ are then checked for hierarchical arc-consistency w.r.t. C.

3. **If** the domain of some variable becomes empty, an inconsistency has been detected and the relaxation value assignment σ_r is returned as the hierarchical arc-consistent value assignment for the (relaxed) HCSP.
   Otherwise, if the domain of some variable X has been restricted due to the application of some constraint, all other constraints $C_1, \ldots, C_m$ connected with X have to be revised again: $Q \leftarrow Q \cup \{C_1, \ldots, C_m\}$

4. **If** the constraint queue Q is not empty, the process continues with step 2.
   Otherwise, the current value assignment σ is hierarchical arc-consistent and is stored as new relaxation value assignment σ_r.

5. **If** there are no more constraints that can be strengthened, i.e. all constraints and no relaxation tuples are currently taken into account, σ_r is returned as the value assignment satisfying the entire CSP.
   Otherwise, the CSP is strengthened in some aspect, i.e. some set of relaxation tuples is removed or some constraint is added that has not yet been taken into account. The constraint queue Q is set to hold the newly strengthened constraints and the process continues with step 2.

Step 2 contains the very heart of the constraint *propagation* algorithm, which is how to select the next constraint from Q to revise. Here, a set of heuristics are used, for example, to prefer the constraint with the mostly restricted domains of its variables.

Step 5 incorporates the very heart of the *relaxation* procedure, i.e. the selection of the next strengthening step. Again, this selection is also based on heuristics guided by the discrete weights of the relaxation tuples and the constraints themselves.

4.3.6 Implementation issues

The CONTAX system has been implemented in Common Lisp and runs on a variety of hardware platforms including Symbolics UX1200 Lisp boards. In order to develop an open system which can easily be extended towards other types of domains, the mechanisms like dispatching on type and the use of generic functions as provided by the Common Lisp Object System (CLOS) have been used intensively.

Besides being integrated in COLAB, there also exists a stand-alone version of CONTAX with a programmer interface to LISP [Meyer and Steinle, 1992] that enables any LISP-based application to use CONTAX as its own constraint-reasoning formalism.

5 Using CONTAX for lathe-tool selection

The constraint system CONTAX presented so far has been used to formalize and solve the lathe-tool selection problem as it occurs in a real-life process planning application. The principal approach how to formalize the lathe-tool selection problem as a CSP has been given before using a small example. However, in our real-life application we have to deal with a considerably larger set of constraints on some more variables ranging over much larger domains.

The current implementation of the lathe-tool selection module [Tolzmann, 1992] covers 59 different cutting plates, 62 holders, and 22 different materials. The

problem constraints determining the tool selection include technological 'rules' like

> **If** the workpiece is stable (a criterion depending on the length/diameter ratio) and the `edge-angle` is of class `large` or `medium`,
> **then** prefer a medium tool-cutting edge-angle (`tc-edge-angle`)

as well as economical ones like

> **If** the current process is rough-turning the workpiece,
> **then** a quadratic plate should be preferred to a triangular one wherever possible. Quadratic plates can be used more often in a process since they have one more tip.

Due to the compact notation of concepts (or classes) representing subsets of the domains, together with the ability to define constraints over abstract concepts and making use of inheriting properties from superconcepts to subconcepts, the increasing domain sizes of more than 400 elements in total poses no real efficiency problems. Since variables and constraints are compiled into CLOS objects, the most time-consuming task has been the generation of the constraint network. Thus, as in our concrete application all constraints and variables that have to be generated are known in advance, the constraint net can be built up when loading the whole planning system; the constraint-solving process itself can then be performed very efficiently.

Moreover, as expected, the problem formulation acquired from human experts as well as from text-books turned out to be contradictive. Thus, constraints had to be weighted in order to cope with such implicit contradictions. The ability of CONTAX to relax the problem step by step until it becomes solvable, has been essential for the success of our approach. Trying to formulate the problem using other constraint systems, would have required a complete problem reformulation.

6 Conclusion

In this paper we have shown how the notion of hierarchical arc-consistency introduced by [Mackworth *et al.*, 1985] can serve as a basis for developing a constraint system over hierarchically structured finite domains.

The application of lathe-tool selection motivated the development of a constraint relaxation procedure, which enables the system to cope with over-specified constraint problems.

Both extensions of classical constraint systems, exploiting hierarchically structured domains and incorporating constraint relaxation methods, have been essential for our approach to use constraint propagation for lathe-tool selection in a real-life application.

Acknowledgements

The research presented in this paper has been carried out within the ARC-TEC project and was supported by BMFT under grant ITW 8902 C4.

I would like to thank Christoph Jakfeld and Frank Steinle who mainly implemented the CONTAX system, Jörg Müller for formalizing the lathe-tool selection problem as CSP and realizing a first prototype, and Enno Tolzmann for implementing the lathe-tool selection module of the ARC-TEC planning system with CONTAX. The three anonymous referees also provided useful comments on the presentation of the paper.

References

[Baader and Hanschke, 1991] 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.

[Bernardi *et al.*, 1991] A. Bernardi, H. Boley, K. Hinkelmann, P. Hanschke, C. Klauck, O. Kühn, R. Legleitner, M. Meyer, M.M. Richter, G. Schmidt, F. Schmalhofer, and W. Sommer. ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge. In *Expert Systems and their Applications: Tools, Techniques and Methods*, Avignon, France, 1991. Also available as Research Report RR-91-27, DFKI GmbH.

[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, Reisensburg, Germany, September 1991.

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

[Brachman and Schmolze, 1985] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Descotte and Latombe, 1985] Y. Descotte and J.-C. Latombe. Making Compromises among Antagonist Constraints in a Planner. *Artifical Intelligence*, 27:183–217, 1985.

[Freuder, 1978] E. Freuder. Synthesizing Constraint Expressions. *Communications of the ACM*, 21(11):958–966, 1978.

[Hein and Meyer, 1992] H.-G. Hein and M. Meyer. A WAM Compilation Scheme. In A. Voronkov, editor, *Logic Programming: Proceedings of the* 1^{st} *and* 2^{nd} *Russian Conferences*, volume 592 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 201–214. Springer-Verlag, Berlin, Heidelberg, 1992.

[Hinkelmann, 1992] K. Hinkelmann. Forward Logic Evaluation: Compiling a Partially Evaluated Meta-interpreter into the WAM. In *Proceedings German Workshop on Artificial Intelligence, GWAI-92*. Springer-Verlag, Berlin, Heidelberg, September 1992.

[Klauck *et al.*, 1991] C. Klauck, A. Bernardi, and R. Legleitner. FEAT-REP: Representing features in CAD/CAM. In *4th International Symposium on Artificial Intelligence: Applications in Informatics*, Cancun, Mexiko, 1991. An extended Version is also available as Research Report RR-91-20, DFKI GmbH.

[Kusiak, 1988] A. Kusiak, editor. *Artificial Intelligence Implications for CIM.* IFS Publications, Bedford and Springer-Verlag, Berlin, Heidelberg, 1988.

[Mackworth and Freuder, 1985] A. K. Mackworth and E. C. Freuder. The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 25:65–73, 1985.

[Mackworth *et al.*, 1985] A. Mackworth, J. Mulder, and W. Havens. Hierarchical Arc Consistency: Exploiting Structured Domains in Constraint Satisfaction Problems. *Computational Intelligence*, 1:118–126, 1985.

[Mackworth, 1977] A. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[Meseguer, 1989] P. Meseguer. Constraint Satisfaction Problems: An Overview. *AI Communications*, 2(1):3–17, 1989.

[Meyer and Steinle, 1992] M. Meyer and F. Steinle. *CONTAX Programmer's Guide.* DFKI GmbH, P. O. Box 2080, Kaiserslautern, Germany, 1992.

[Meyer *et al.*, 1992] M. Meyer, H.-G. Hein, and J. Müller. FIDO: Finite Domain Consistency Techniques in Logic Programming. In A. Voronkov, editor, *Logic Programming: Proceedings of the* 1^{st} *and* 2^{nd} *Russian Conferences*, volume 592 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 294–301. Springer-Verlag, Berlin, Heidelberg, 1992.

[Schmalhofer *et al.*, 1991] F. Schmalhofer, O. Kühn, and G. Schmidt. Integrated Knowledge Acquisition from Text, Previously Solved Cases, and Expert Memories. *Applied Artificial Intelligence*, 5:311–337, 1991.

[Tolzmann, 1992] E. Tolzmann. Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX. DFKI GmbH, P. O. Box 20 80, 6750 Kaiserslautern, Germany, July 1992. In German.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-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 published 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-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
- A Logic-Based Approach -
17 pages

RR-91-27
A. Bernardi, H. Boley, Ph. Hanschke, K. Hinkelmann, Ch. Klauck, O. Kühn, R. Legleitner, M. Meyer, M. M. Richter, F. Schmalhofer, G. Schmidt, W. Sommer:
ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge
18 pages

RR-91-28
Rolf Backofen, Harald Trost, Hans Uszkoreit:
Linking Typed Feature Formalisms and Terminological Knowledge Representation Languages in Natural Language Front-Ends
11 pages

RR-91-29
Hans Uszkoreit: Strategies for Adding Control Information to Declarative Grammars
17 pages

RR-91-30
Dan Flickinger, John Nerbonne:
Inheritance and Complementation: A Case Study of *Easy* Adjectives and Related Nouns
39 pages

RR-91-31
H.-U. Krieger, J. Nerbonne:
Feature-Based Inheritance Networks for Computational Lexicons
11 pages

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-91-34
Bernhard Nebel, Christer Bäckström:
On the Computational Complexity of Temporal Projection and some related Problems
35 pages

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

RR-92-01
Werner Nutt: Unification in Monoidal Theories is Solving Linear Equations over Semirings
57 pages

RR-92-02
Andreas Dengel, Rainer Bleisinger, Rainer Hoch, Frank Hönes, Frank Fein, Michael Malburg:
Π_{ODA}: The Paper Interface to ODA
53 pages

RR-92-03
Harold Boley:
Extended Logic-plus-Functional Programming
28 pages

RR-92-04
John Nerbonne: Feature-Based Lexicons:
An Example and a Comparison to DATR
15 pages

RR-92-05
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Michael Schulte, Rainer Stark:
Feature based Integration of CAD and CAPP
19 pages

RR-92-06
Achim Schupetea: Main Topics of DAI: A Review
38 pages

RR-92-07
Michael Beetz:
Decision-theoretic Transformational Planning
22 pages

RR-92-08
Gabriele Merziger: Approaches to Abductive Reasoning - An Overview -
46 pages

RR-92-09
Winfried Graf, Markus A. Thies:
Perspektiven zur Kombination von automatischem Animationsdesign und planbasierter Hilfe
15 Seiten

RR-92-10
M. Bauer: An Interval-based Temporal Logic in a Multivalued Setting
17 pages

RR-92-11
Susane Biundo, Dietmar Dengler, Jana Koehler:
Deductive Planning and Plan Reuse in a Command Language Environment
13 pages

RR-92-13
Markus A. Thies, Frank Berger:
Planbasierte graphische Hilfe in objektorientierten Benutzungsoberflächen
13 Seiten

RR-92-14
Intelligent User Support in Graphical User Interfaces:
1. InCome: A System to Navigate through Interactions and Plans
   Thomas Fehrle, Markus A. Thies
2. Plan-Based Graphical Help in Object-Oriented User Interfaces
   Markus A. Thies, Frank Berger

22 pages

RR-92-15
Winfried Graf: Constraint-Based Graphical Layout of Multimodal Presentations
23 pages

RR-92-16
Jochen Heinsohn, Daniel Kudenko, Berhard Nebel, Hans-Jürgen Profitlich: An Empirical Analysis of Terminological Representation Systems
38 pages

RR-92-17
Hassan Aït-Kaci, Andreas Podelski, Gert Smolka:
A Feature-based Constraint System for Logic Programming with Entailment
23 pages

RR-92-18
John Nerbonne: Constraint-Based Semantics
21 pages

RR-92-19
Ralf Legleitner, Ansgar Bernardi, Christoph Klauck
PIM: Planning In Manufacturing using Skeletal Plans and Features
17 pages

RR-92-20
John Nerbonne: Representing Grammar, Meaning and Knowledge
18 pages

RR-92-21
Jörg-Peter Mohren, Jürgen Müller
Representing Spatial Relations (Part II) -The Geometrical Approach
25 pages

RR-92-22
Jörg Würtz: Unifying Cycles
24 pages

RR-92-23
Gert Smolka, Ralf Treinen:
Records for Logic Programming
38 pages

RR-92-24
Gabriele Schmidt: Knowledge Acquisition from Text in a Complex Domain
20 pages

RR-92-25
Franz Schmalhofer, Ralf Bergmann, Otto Kühn, Gabriele Schmidt: Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations
12 pages

RR-92-26
Franz Schmalhofer, Thomas Reinartz, Bidjan Tschaitschian: Intelligent documentation as a catalyst for developing cooperative knowledge-based systems
16 pages

RR-92-27
Franz Schmalhofer, Jörg Thoben: The model-based construction of a case-oriented expert system
18 pages

RR-92-29
Zhaohur Wu, Ansgar Bernardi, Christoph Klauck: Skeletel Plans Reuse: A Restricted Conceptual Graph Classification Approach
13 pages

RR-92-33
Franz Baader
Unification Theory
22 pages

RR-92-34
Philipp Hanschke
Terminological Reasoning and Partial Inductive Definitions
23 pages

RR-92-35
Manfred Meyer
Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment
18 pages

RR-92-36
Franz Baader, Philipp Hanschke
Extensions of Concept Languages for a Mechanical Engineering Application
15 pages

RR-92-37
Philipp Hanschke
Specifying Role Interaction in Concept Languages
26 pages

RR-92-38
Philipp Hanschke, Manfred Meyer
An Alternative to H-Subsumption Based on Terminological Reasoning
9 pages

DFKI Technical Memos

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 from a Partially Evaluated Meta Interpreter
16 pages

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

TM-91-15
Stefan Bussmann: Prototypical Concept Formation
An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang:
Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh
A Cognitiv Analysis of Event Structure
21 pages

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

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

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

DFKI Documents

D-91-17
Andreas Becker:
Analyse der Planungsverfahren der KI im Hinblick auf ihre Eignung für die Abeitsplanung
86 Seiten

D-91-18
Thomas Reinartz: Definition von Problemklassen im Maschinenbau als eine Begriffsbildungsaufgabe
107 Seiten

D-91-19
Peter Wazinski: Objektlokalisation in graphischen Darstellungen
110 Seiten

D-92-01
Stefan Bussmann: Simulation Environment for Multi-Agent Worlds - Benutzeranleitung
50 Seiten

D-92-02
Wolfgang Maaß: Constraint-basierte Plazierung in multimodalen Dokumenten am Beispiel des Layout-Managers in WIP
111 Seiten

D-92-03
Wolfgan Maaß, Thomas Schiffmann, Dudung Soetopo, Winfried Graf: LAYLAB: Ein System zur automatischen Plazierung von Text-Bild-Kombinationen in multimodalen Dokumenten
41 Seiten

D-92-04
Judith Klein, Ludwig Dickmann: DiTo-Datenbank - Datendokumentation zu Verbrektion und Koordination
55 Seiten

D-92-06
Hans Werner Höper: Systematik zur Beschreibung von Werkstücken in der Terminologie der Featuresprache
392 Seiten

D-92-07
Susanne Biundo, Franz Schmalhofer (Eds.):
Proceedings of the DFKI Workshop on Planning
65 pages

D-92-08
Jochen Heinsohn, Bernhard Hollunder (Eds.): DFKI Workshop on Taxonomic Reasoning Proceedings
56 pages

D-92-09
Gernod P. Laufkötter: Implementierungsmöglichkeiten der integrativen Wissensakquisitionsmethode des ARC-TEC-Projektes
86 Seiten

D-92-10
Jakob Mauss: Ein heuristisch gesteuerter Chart-Parser für attributierte Graph-Grammatiken
87 Seiten

D-92-11
Kerstin Becker: Möglichkeiten der Wissensmodellierung für technische Diagnose-Expertensysteme
92 Seiten

D-92-12
Otto Kühn, Franz Schmalhofer, Gabriele Schmidt:
Integrated Knowledge Acquisition for Lathe Production Planning: a Picture Gallery (Integrierte Wissensakquisition zur Fertigungsplanung für Drehteile: eine Bildergalerie)
27 pages

D-92-13
Holger Peine: An Investigation of the Applicability of Terminological Reasoning to Application-Independent Software-Analysis
55 pages

D-92-15
DFKI Wissenschaftlich-Technischer Jahresbericht 1991
130 Seiten

D-92-16
Judith Engelkamp (Hrsg.): Verzeichnis von Softwarekomponenten für natürlichsprachliche Systeme
189 Seiten

D-92-17
Elisabeth André, Robin Cohen, Winfried Graf, Bob Kass, Cécile Paris, Wolfgang Wahlster (Eds.):
UM92: Third International Workshop on User Modeling, Proceedings
254 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-92-18
Klaus Becker: Verfahren der automatisierten Diagnose technischer Systeme
109 Seiten

D-92-19
Stefan Dittrich, Rainer Hoch: Automatische, Deskriptor-basierte Unterstützung der Dokumentanalyse zur Fokussierung und Klassifizierung von Geschäftsbriefen
107 Seiten

D-92-21
Anne Schauder: Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars
57 pages

Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment

Manfred Meyer

RR-92-35
Research Report