Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

**Document**
D-92-26

# Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX

Enno Tolzmann

September 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

# Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX

**Enno Tolzmann**

DFKI-D-92-26

Diese Arbeit wurde finanziell unterstützt durch das Bundesministerium für Forschung und Technologie (FKZ ITW-8902 C4).

# Inhaltsverzeichnis

# 1 Einleitung

Das Modul *werkzeug-konfig* dient zur Werkzeugauswahl bei der Drehbearbeitung mit Schneidstoffen aus Hartmetall aufgrund gegebener Geometrie- und Technologieinformationen. Es wurde mit dem Constraint-System CONTAX und der Programmiersprache Lisp implementiert.

Dabei soll das Programm nicht nur „gute Ergebnisse“ mit „akzeptablen Antwortzeiten“ liefern, sondern auch ein hohes Maß an Änderungsfreundlichkeit und Flexibilität bieten. Die Ergebnisgüte bei gegebenen Eingabedaten hängt vor allem von der Qualität der formulierten Constraints und ihrer Gewichtung ab, sowie von dem CONTAX zugrundeliegenden Inferenzmechanismus. Die Antwortzeit hängt vorwiegend von der Anzahl und der Komplexität der definierten Constraints, sowie von dem Laufzeitverhalten des Constraint-Systems ab. Flexibilität und Änderungsfreundlichkeit des Programms werden durch die bei Expertensystemen übliche weitgehende Trennung von anwendungsspezifischem Wissen und dem Inferenzmechanismus durch den Einsatz von CONTAX ermöglicht. Die definierten Lisp-Funktionen werden lediglich als Benutzerschnittstellen und zur Datenverwaltung eingesetzt.

Die folgenden Kapitel sind so aufgebaut, daß sie den Ansprüchen verschiedener Benutzergruppen genügen. Für den Benutzer, der lediglich eine kurze Einführung in die Bedienung des Programmes wünscht, ist die Funktionsbeschreibung in Kapitel 4 gedacht. Falls sich der Benutzer weiter über die Grundlagen der Implementierung und die Leistungsfähigkeit der Werkzeugauswahl informieren möchte, so sollte er die Kapitel 2 und 6 lesen. Um die interne Funktionsweise des Programmes zu verstehen und Wartungsarbeiten durchführen zu könen, ist es zusätzlich erforderlich, das Kapitel 3 über das Constraint-System CONTAX und das Kapitel 5, welches spezielle Implementierungsfragen erläutert, zu lesen.

# 2 Grundlagen der Implementierung

Vor der Implementierung des Werkzeugauswahlmoduls stand die Problemformulierung. Diese ließ sich in vier aufeinanderfolgende Teilprobleme gliedern:

1. Festlegung des Einsatzbereiches des Werkzeugauswahlmoduls.
2. Spezifikation der Eingabe- und der Ausgabedaten.
3. Wahl der Wissensrepräsentation.
4. Formalisierung des Expertenwissens.

Als strukturierte Vorgehensweise zur Problemcharakterisierung wurde die Interviewtechnik verwendet, die zum Wissenstransfer vom Experten zum Wissensingenieur dient (Puppe [6]). Dabei führten wir vorwiegend unstrukturierte Interviews und Introspektionen durch.

## 2.1 Einsatzbereich

Vorausgeschickt werden muß, daß es nötig war, den Einsatzbereich des Werkzeugauswahlmoduls stark einzuschränken, um angemessene Antwortzeiten zu ermöglichen und den Implementierungsaufwand zu begrenzen. Deshalb beschränkt sich *werkzeug-konfig* auf die Auswahl von Standardwerkzeug bei der Drehbearbeitung mit Schneidplatten aus Hartmetall. Weitere Einschränkungen sind:

1. Es werden lediglich Werkzeuge für die Außenbearbeitung ausgewählt.
2. Bei der Bearbeitungsart wird ausschließlich zwischen Schruppen und Schlichten unterschieden.
3. Spezialwerkzeuge zum Gewinde- und Nutendrehen, sowie Abstechwerkzeuge können nicht ausgewählt werden. Dieses gilt ebenfalls für Einstichwerkzeuge.

Die ersten beiden Einschränkungen lassen sich relativ einfach durch Anpassung des Definitionsbereiches der CONTAX-Variablen aufheben. Hingegen wäre es zum Aufheben der dritten Einschränkung nötig, das gesamte Ein-/Ausgabekonzept von *werkzeug-konfig* zu ändern.

## 2.2 Spezifikation der Ein-/Ausgabedaten

Als Ausgangspunkt für die Spezifikation der Eingabe- und der Ausgabedaten dienen die infragekommenden Drehwerkzeuge (siehe Abschnitt 2.1). Mit dem standardisierten Bezeichnungssystem nach ISO 5608 und 1832 lassen sich Drehwerkzeuge, bestehend aus Klemmhalter und Wendeschneidplatte, nach dem Prinzip des Baukastensystems konfigurieren.

Die konfigurierten Drehwerkzeuge bilden die Ausgabedaten und werden durch die in Tabelle 1 aufgeführten Attribute und ihre Ausprägungen, die eine Teilmenge des ISO Bezeichungssystems bilden, eindeutig festgelgt. Die Klemmhalter und Wendeschneidplatten, entnommen dem Katalog der Firma SANDVIK [7], bilden zusammen mit dem Schneidstoff den statischen Teil der Eingabedaten. Die verfügbaren Klemmhalter und Wendeschneidplatten sind in zwei Dateien gespeichert und werden durch Lisp-Funktionen verwaltet. Die möglichen Ausprägungen des Schneidstoffes werden durch den gleichnamigen CONTAX-Datentyp (siehe Abschnitt 2.3) spezifiziert.

Als dynamische Eingabedaten sind eine Anzahl von Geometrie- und Technologieinformationen erforderlich, die es ermöglichen, eine sinnvolle Werkzeugauswahl durchzuführen. Dazu wird einerseits eine Beschreibung des zu erzeugenden Werkstücks (Zielwerkstück) benötigt, die sowohl

| Attribut | Klemmhalter | Schneidplatte | Attributsausprägung |
|---|---|---|---|
| Werkzeugsystem | x | x | TMAXU, TMAXP |
| Spannsystem | x | | C, P, M, S |
| Plattenform | x | x | R, S, T, C, D |
| Halterform | x | | B, D, E, F, G, J K ,L ,R ,S ,T ,V |
| Vorschubrichtung | x | | N, R+L (R und L) |
| Plattengröße | x | x | von 05 bis 32 nach ISO-Kode |
| Schafthöhe | x | | Angaben in cm |
| Schaftbreite | x | | Angaben in cm |
| Werkzeuglänge | x | | von 60 bis 300 cm nach ISO-Kode |
| Freiwinkel | | x | 0 |
| Toleranzen | | x | M |
| Leitstufe | | x | F, N, R, A, G, M |
| Plattendicke | | x | Angaben in cm nach ISO-Kode |
| Eckenradius | | x | Angaben in cm nach ISO-Kode |
| Schneidstoffe | | | CT515, GC424, GC3015 |

Tabelle 1: statische Eingabedaten / Ausgabedaten

| Attribut | Attributsausprägung |
|---|---|
| Zielwerkstück | als TEC-REP-Datei repräsentiert |
| Werkstückstoff | Baustahl, niedriglegierter Stahl, Gußeisen |
| Werkstückstabilität | labil, stabil |
| Bearbeitungsart | Schruppen, Schlichten |
| Bearbeitungsrichtung | Längsdrehen (von links oder rechts), Plandrehen |

Tabelle 2: dynamische Eingabedaten

geometrische als auch technologische Elemente enthält, andererseits werden als weitere Technologieinformationen die Bearbeitungsart und die Bearbeitungsrichtung benötigt. Die Repräsentation des Zielwerkstücks erfolgt in TEC-REP [1], wobei zusätzlich noch der Werkstückstoff und die Werkstückstabilität angegeben werden müssen. Alle benötigten dynamischen Eingabedaten werden noch einmal in der Tabelle 2 zusammengefaßt. Zusätzlich besteht die Möglichkeit, die Attribute der zu konfigurierenden Drehwerkzeuge bereits bei der Eingabe auf bestimmte Attributsausprägungen einzuschränken, wodurch die Konfiguration von unerwünschten Werkzeugen verhindert wird.

## 2.3 Wahl der Wissensrepräsentation

Die Wahl der Wissensrepräsentation hängt im wesentlichen vom Typ des zu lösenden Problems ab. Bei der Werkzeugkonfiguration handelt es sich um ein Auswahlproblem. Dabei werden vom Experten bestimmte Anforderungen an die Werkzeugauswahl gestellt, deren Erfüllung einen unterschiedlichen Stellenwert haben kann. Die Anforderungen können sowohl technologische als auch ökonomische Aspekte der Werkzeugkonfiguration betreffen, z. B.:

- Falls das Werkstück stabil ist und ein großer oder mittlerer Eckenradius für die Schneidplatte gewählt wurde, dann wähle möglichst einen mittleren Einstellwinkel.
- Bei der Durchführung einer Grobbearbeitung wähle eher eine quadratische Plattenform als eine dreieckige Plattenform.

Die erste der beiden aufgeführten Regeln dient dazu, die auf dem Drehwerkzeug lastenden Kräfte möglichst gering zu halten. Somit handelt es sich in diesem Fall um eine technologische Anforderung. Die zweite Regel ist ökonomischer Natur, da sie versucht, die Schneidplatte auszuwählen, die mehr Schneidkanten besitzt und somit auch häufiger verwendbar ist.

Als eine abstrakte Beschreibung für dieses rein sprachlich formulierte Expertenwissen bieten sich Constraints an, die zur Repräsentation von Relationen dienen (es werden beliebige Beziehungen zwischen Variablen ausgedrückt). Durch eine Gewichtung der Constraints besteht die Möglichkeit, die unterschiedliche Relevanz des sprachlich formulierten Expertenwissens bei der Werkzeugauswahl zu berücksichtigen. Desweiteren erleichtert die Definition einer taxonomischen Hierarchie auf den verwendeten Attributen aus CONTAX-Datentypen die Formulierung von Constraints. Beispielsweise kann das Attribut Eckenradius die Ausprägungen kleiner, mittlerer oder großer Eckenradius annehmen, die wiederum eine Menge von konkreten Werteangaben umfassen.

Alle diese Möglichkeiten bietet das Constraint-System CONTAX, welches aus diesem Grund als Werkzeug für die Wissensrepräsentation verwendet wird. Nähere Informationen dazu können dem CONTAX User Guide [8] entnommen werden.

## 2.4 Formalisierung des Expertenwissens

Wie bereits in Abschnitt 2.3 angesprochen, bestand die Aufgabe bei der Formalisierung des Expertenwissens in der Formulierung von Anforderungen an die Werkzeugauswahl und deren Überführung in gewichtete CONTAX-Constraints. Hierzu stellt CONTAX mehrere Constrainttypen und insgesamt fünf Gewichtsstufen, die von „hard“ bis „soft“ reichen, zur Verfügung. Harte Constraints müssen bei der Werkzeugauswahl immer erfüllt werden. Gelingt dieses nicht, so läßt sich für die gewählten Eingabedaten kein Werkzeug konfigurieren. Constraints mit geringeren Gewichten können (müssen aber nicht) zur weiteren Einschränkung der Werkzeugauswahl herangezogen werden. Die Gewichtsbehandlung des CONTAX-Inferenzmechanismus wird in Abschnitt 3.3 besprochen. Die Formalisierung des Expertenwissens erfolgte in mehreren Schritten.

Im ersten Schritt wurden, ausgehend von dem verfügbaren Eingabedaten, vom Experten Bedingungen formuliert, die bei der Werkzeugauswahl zu beachten sind und somit zur Einschränkung der Ausgabedaten führen. Zwei typische Beispiele für Bedingungen wurden bereits in Abschnitt 2.3 vorgestellt. Es wurden jedoch nur solche Bedingungen akzeptiert, die sich technologisch oder ökonomisch begründen ließen, wodurch die Aufnahme von nicht verifizierbaren Bedingungen in die Menge der Anforderungen vermieden wurde. Außerdem wurden bei sämtlichen Bedingungen die Attributsausprägungen mit abstrakten Datentypen beschrieben. Dieses ermöglichte einerseits, die Bedingungen sehr knapp und verständlich zu formulieren, andererseits konnte die Definition der abstrakten Datentypen in einem späteren Schritt erfolgen. Die Bedingungen wurden anschließend unter Verwendung der abstrakten Datentypen in Constraints überführt. Der Transformationsaufwand war dabei sehr gering, da die Bedingungen und die Constraints nur geringe Unterschiede

bezüglich ihrer Struktur aufwiesen. Alle definierten Constraints sind, mit Kommentaren versehen, im Anhang B aufgeführt.

Im zweiten Schritt wurden die abstrakten Datentypen zur Definition von CONTAX-Variablen verwendet. Die abstrakten Datentypen fassen dabei Attributsausprägungen mit gleichen Eigenschaften zu Klassen zusammen (z.B. $großerEckenwinkel = \{80°, 90°\}$). Die Aufgabe des Experten bestand darin, im Einklang mit seinen formulierten Bedingungen, jedem abstrakten Datentyp eine Menge von Attributsausprägungen zuzuordnen. Die CONTAX-Datentypen sind im Anhang A aufgeführt.

Im dritten Schritt mußte schließlich noch die Gewichtung der Constraints durchgeführt werden. Begonnen wurde dabei mit der Festlegung der harten Constraints durch den Experten. Als nächstes wurden die Constraints ausgewählt, die die nächstniedrigere Gewichtsstufe erhalten sollten. Dieses Verfahren wurde solange fortgesetzt, bis der Experte sämtliche Constraints mit Gewichten versehen hatte.

Bei der Durchführung der drei genannten Schritte zur Formalisierung des Expertenwissens besteht das wesentliche Problem in der Gewichtung der Constraints. Dieses liegt daran, daß die komprimierte Wissensrepräsentation in Form von Gewichten vom Experten verlangt, sein Wissen adäquat in diese alternative Wissensrepräsentation umzuwandeln [2]. Dieses — so hat sich in unserem Fall gezeigt — ist jedoch insbesondere für alle Gewichtsstufen unterhalb von „hard“ nur schwer möglich. Somit war es erforderlich, an Hand von Testläufen festzustellen, welche Gewichte der Veränderung bedürfen und diese entsprechend anzupassen. Da dieses Verfahren bei einer großen Menge von Constraints und vielfältigen Testdaten sehr aufwendig ist und häufig nur geringfügige Verbesserungen bringt, kann es nicht als eine Alternative zu einer durchdachten Gewichtsvergabe in Schritt drei angesehen werden. Dieses könnte beispielsweise durch strukturierte Interviews mit mehreren Experten erreicht werden. Die unterschiedlichen Ergebnisse dieser Interviews könnten dann zu einer Vereinheitlichung der Gewichtung benutzt werden.

## 2.5 Programmaufbau

Bei dem Werkzeugkonfigurationsmodul *w-konfig* handelt es sich um ein wissenbasiertes System, das sich durch die Trennung von Wissen und Inferenzmechanismus auszeichnet. Es besteht im wesentlichen aus den drei Komponenten Wissensbasis, Datenbasis und Inferenzmechanismus. Zusätzlich sind noch die in Lisp geschriebenen Benutzerfunktionen zu erwähnen, die den Komponenten nicht eindeutig zugeordnet werden können. Dieses liegt daran, daß bei ihrer Implementierung die Trennung von Wissen und Inferenzmechanismus zugunsten einer geringeren Antwortzeit aufgegeben wurde (siehe Abschnitt 6.2). Bei der nun folgenden Beschreibung der Komponenten sollen deshalb die Benutzerfunktionen nicht in die Betrachtung eingeschlossen werden. Die Wissensbasis besteht aus den 21 definierten, gewichteten Constraints. Diese werden, abhängig von den dynamischen Eingabedaten, zum Aufbau eines Constraint-Netzes verwendet.

Zur Datenbasis gehören die statischen und dynamischen Eingabedaten (siehe Abschnitt 2.2). Die dynamischen Eingabedaten werden bei jedem Programmaufruf vom Benutzer übergeben, die statischen Eingabedatei liegen hingegen in Dateien des Werkzeugauswahlmoduls vor. Die statischen Eingabedaten bestehen aus

der Datei *Platten.data*, die 59 Schneidplatten enthält,

der Datei *Werkzeug.data*, die 62 Drehwerkzeuge enthält sowie

den CONTAX-Datentypen in der Datei *constraint.lisp*, die neben der Spezifikation der 3 Schneidstoffe auch die Spezifikationen für alle anderen Eingabedaten enthalten.

Insgesamt wurden 22 CONTAX-Datentypen definiert, die 117 Attributsausprägungen enthalten, ohne die abstrakte Datentypen mitzuzählen.

Der Inferenzmechanismus besteht aus der Inferenzmaschine des CONTAX-Systems, die die Propagierung des aufgebauten Constraint-Netzes unter Berücksichtigung der unterschiedlichen Gewichtung der enthaltenen Constraints durchführt.

# 3 Das Constraint-System CONTAX

Das CONTAX-System, ein Teil der Knowledge Compilation Shell COLAB, die am DFKI entwickelt wurde und aus Vorwärts- und Rückwärtsregeln, taxonomischen Hierarchien und Constraints besteht, ist ein universelles System zum Lösen von Constraints.

## 3.1 Constraints und taxonomische Hierarchien

Ein Constraint besteht aus einer Menge von Variablen und einer Relation über diesen Variablen. Es repräsentiert somit, im Gegensatz zu Regeln, ungerichtete Zusammenhänge zwischen Variablen (z.B. eine Gleichung). CONTAX läßt nur Variablen mit diskreten Wertebereichen zu. Zusätzlich können taxonomische Hierarchien (isa-hierarchies) auf den Wertebereichen als CONTAX-Datentypen definiert werden und somit Objekte mit gleichen Eigenschaften zu Klassen zusammengefaßt werden.

CONTAX stellt dem Benutzer zwei Möglichkeiten zur Verfügung, Constraints zu definieren. Bei primitiven Constraints werden alle Tupel, die Elemente der zu definierenden Relationen sind, aufgezählt. Falls möglich können dabei die auf den Wertebereichen der Variablen definierten hierarchischen Strukturen zur abkürzenden Schreibweise verwendet werden. Bei Prädikat-Constriants wird eine benutzerdefinierte Lisp-Funktion als Prädikat verwendet, die bei der Auswertung des entsprechenden Constraints evaluiert wird. CONTAX macht allerdings die Einschränkung, daß die Lisp-Funktion auf den Blattknoten der taxomomischen Hierarchien der Variablen operieren muß. Prädikat-Constraints eignen sich inbesondere zur Darstellung von Relationen, die nicht ausschließlich auf diskreten Wertebereichen (z.B. $<$) definiert sind. Die definierten Constraints lassen sich zu einem Constraint-Netz zusammenfassen, das eine Menge von Relationen und eine Menge von Variablen, über denen die Relationen definiert sind, darstellt.

## 3.2 Propagierung

Vor der Propagierung können die verwendeten Variablen bereits auf eine Unterbereich ihrer Wertemenge, die im Normalfall den gesamten Wertebereich umfaßt, eingeschränkt werden. Die Propagierung besteht im wesentlichen darin, daß die Beschränkung der Wertemenge einer Variablen über die mit ihr verbundenen Constraints an andere Variablen weitergegeben wird, bis keine weiteren Einschränkungen von Variablen mehr möglich sind oder eine Variable die leere Menge enthält (Inkonsistenz). CONTAX erlaubt sowohl die lokale, als auch die globale Propagierung eines Constraint-Netzes. Bei der lokalen Propagierung wird eine kanten-konsitente Lösung (*arc-consitency*) in Form einer Lösungsmenge für jede betroffene Variable berechnet, d.h. für jedes Paar ($X_i$,$Y_j$) von Variablen existiert zu jedem Element der Lösungsmenge von $X_i$ ein Element der Lösungsmenge von $Y_j$, das alle Constraints zwischen $X_i$ und $Y_j$ erfüllt. Bei der globalen Propagierung werden Lösungstupel berechnet, die alle im Constraint-Netz enthaltenen Constraints simultan erfüllen. Jede global konsistente Lösung ist demzufolge auch eine lokal konsistente Lösung aber nicht umgekehrt. Da die lokale Propagierung eine deutlich niedrigere Zeitkomplexität als die globale Propagierung hat, wird sie von CONTAX bei der globalen Propagierung zur Reduzierung des Suchraumes verwendet.

## 3.3 Gewichtsbehandlung

Eine weitere Möglichkeit des CONTAX-Systems besteht in der Vergabe von Gewichten an die definierten Constraints, die, falls erwünscht, bei der Propagierung berücksichtigt werden. Die Gewichte umfassen dabei die insgesamt fünf Gewichtsstufen „hard“, „strong“, „medium“, „weak“ und „soft“. Bei der Progagierung kann mit dem keyword „minimum-priority“ angegeben werden, bis zu welcher Gewichtsstufe die definierten Constraints berücksichtigt werden sollen. Die Propagierung erfolgt in

diesem Fall, entsprechend der minimum-priority, in mehreren Stufen. In der ersten Stufe besteht das Constraint-Netz nur aus den harten Constraints. Ist die lokale Propagierung erfolgreich, so wird das Ergebnis abgespeichert und das Constraint-Netz um die Constraints mit dem nächstniedrigeren Gewicht erweitert. Dieser Vorgang stoppt, falls die lokale Propagierung zur Inkonsistenz führt oder das Constraint-Netz, welches alle Constraints bis zur minimum-priority enthält, erfolgreich propagiert wurde. Im ersten Fall wird das Ergebnis, falls möglich, zurückgesetzt. Ist ein Zurücksetzen nicht möglich, so existiert keine konsistente Lösung. Bei der globalen Propagierung wird anschließend noch das letzte Ergebnis der erfolgreichen lokalen Propagierung global propagiert. Sollte dieses zur Inkonsistenz führen, so wird das abgespeicherte Ergebnis der lokalen Propagierung mit dem nächstniedrigeren Gewicht für eine weitere globale Propagierung verwendet. Dieser Vorgang wiederholt sich, bis die globale Propagierung zum Erfolg führt oder die globale Propagierung des Constraint-Netz mit dem Gewicht „hard“ scheitert. Die globale Propagierung mit Gewichtsbehandlung ist zur Zeit noch nicht im Constraint-System CONTAX implementiert.

# 4 Funktionsbeschreibung

In den folgenden vier Abschnitten werden die Dateien, die Installation, die Ausführung und die exportierten Funktionen des Werkzeugauswahlmoduls *werkzeug-konfig* beschrieben. Für den Benutzer, der lediglich mit dem Programm arbeiten möchte ohne eine Anpassung der Attributsausprägungen (CONTAX-Konzepte) oder der Constraints vorzunehmen, ist es ausreichend, dieses Kapitel zu lesen.

## 4.1 Dateien

*werkzeug-konfig* besteht aus folgenden, nicht compilierten Dateien:

| | |
|---|---|
| w_datenbasis.lisp | besteht aus allen Funktionen, die zur Verwaltung der verfügbaren Drehwerkzeuge dienen.<br>Die exportierten Funktionen sind: **eingabe, laden-und-prüfen-htable, ausgabe-htable** |
| constraints.lisp | enthält alle wesentlichen Komponenten der Wissensbasis des Moduls. Die hierarchisch strukturierten Datentypen der CONTAX-Variablen (Konzepte), die CONTAX-Variablen selbst und die Constraints werden definiert. |
| w_auswahl.lisp | enthält die Benutzerfunktionen zur Ausführung der Werkzeugauswahl. Weitere enthaltene Funktionen dienen zur Auswertung der Eingabedaten, Instanzierung der CONTAX-Variablen, Aufbau des Constraint-Netzes und dessen Propagierung.<br>Die exportierten Funktionen sind: **ausgabe-werkstück, init-werkstück, werkzeugauswahl, prop.** |
| Platten.data | enthält die verfügbaren Schneidplatten. |
| Werkzeug.data | enthält die verfügbaren Werkzeughalter. |
| w_defpackage.lisp | definiert das package *werkzeugkonfiguration* mit dem nickname *wzk.* |
| werkzeug_konfig.lisp | lädt alle relevanten Dateien des Werkzeugauswahlmoduls. |

## 4.2 Installation

Alle Dateien des Werzeugauswahlmoduls befinden sich im Verzeichnis */home/tolzmann/Projekt.* Durch das Laden der Datei werkzeug_konfig.lisp wird das CONTAX-System geladen, werden alle relevanten Funktionen des Werkzeugauswahlmoduls in das package *werkzeugkonfiguration* geladen und die externen Symbole (siehe Abschnitt 4.4) zugriffsfähig gemacht, sowie die Dateien Platten.data und Werkzeug.data in Hash-Tabellen geladen.

Vor dem Laden der Dateien Platten.data und Werkzeu.data wird allerdings noch ein Konsistenztest zwischen den in ihnen enthaltenen Daten und den in constraint.lisp definierten Datentypen durchgeführt. Mißlingt dieser Test, so erfolgt der Abbruch des Ladevorgangs und eine Fehlermeldung wird geliefert. Dieses kann allerdings nur geschehen, falls vorher eine fehlerhafte Änderung der CONTAX-Datentypen durchgeführt wurde.

Zu Testzwecken befindet sich zusätzlich die TEC-REP-Datei Testwerkstück.data im obengenannten Verzeichnis. Sollte *werkzeug-konfig* in ein anderes Verzeichnis kopiert werden, so müssen die in der Datei werkzeug_konfig.lisp angegebenen Pfadnamen entsprechend angepaßt werden. Falls sich die Werkstückrepräsentationen in einem anderen Verzeichnis als das Werkzeugauswahlmodul befinden, so besteht die Möglichkeit, mit **set-pathname-werkstueck** einen Verweis auf dieses Verzeichnis zu legen.

## 4.3 Programmausführung

Um erstmals eine Werkzeugauswahl durchzuführen, muß

1. eine Werkstückrepräsentation mit **init-werkstueck** geladen werden,
2. **werkzeugauswahl** mit der Bearbeitungsliste, der Berarbeitungsrichtung, dem Bearbeitungsverfahren und den erforderlichen Instanzierungen von CONTAX-Variablen aufgerufen werden
3. und die Funktion **prop** aufgerufen werden.

Soll eine erneute Werzeugauswahl für das gleiche Werkstück stattfinden, so kann Schritt 1 übersprungen werden.
Falls Werkzeughalter oder Schneidplatten in die Datenbasis eingefügt oder aus der Datenbasis gelöscht werden sollen, so muß dazu **eingabe** aufgerufen werden.

## 4.4 Exportierte Funktionen

**set-pathname-werkstueck** *pathname* [Funktion]
setzt den Pfadnamen zur Lokalisierung des Verzeichnisses, in dem sich die Werkstückrepräsentationen befinden. *pathname* muß dabei vom gleichnamigen Typ sein.

**init-werkstueck** *dateiname werkstückstoff werkstückstabilität* [Funktion]
lädt die aktuelle Werkstückrepräsentation, *werkstückstoff* und *werkstückstabilität*. Die Werkzeugauswahl bezieht sich immer auf die letzte, mit **init-werkstueck** geladene Werkstückrepräsentation.

*dateiname* vom Typ String bezeichnet den Namen einer TEC-REP-Datei ohne die Angabe ihrer Typenbezeichnung, die .data sein muß. *werkstückstoff* und *werkstückstabilität* müssen von den gleichnamigen, in der Datei constraint.lisp definierten Datentyp sein und gequotet übergeben werden.

Bsp.: (init-werkstueck "Testwerkstueck" 'GG20 'stabil)

**werkzeugauswahl** *bearbeitungsliste berarbeitungsrichtung bearbeitungsverfahren* &rest $var_1$ $werteliste_1 \ldots var_n$ $werteliste_n$ [Funktion]
berechnet aus *bearbeitungsliste*, *bearbeitungsrichtung* und der aktuellen Werkstückrepräsentation die Winkelvorgaben, die als Restriktionen für die Werkzeugauswahl dienen. Die CONTAX-Variablen werden abhängig von *bearbeitungsrichtung*, *bearbeitungsverfahren* und dem &rest-Argument instanziert und das Constraint-Netz wird aufgebaut.

*bearbeitungsliste* besteht aus Listen benachbarter Werkstückteile (Identifikationslisten), die durch Nummern eindeutig spezifiziert sind. Diese Identifikationslisten legen die Werkstückteile fest, für deren Bearbeitung ein Drehwerkzeug konfiguriert werden soll. Die Angabe der Werkstueckteile muß dabei in jeder Identifikationsliste von links nach rechts erfolgen. *bearbeitungsrichtung* muß ein gequoteter Ausdruck sein, der folgende Werte annehmen darf:

r Bearbeitung von rechts nach links.

l Bearbeitung von links nach rechts.

v Längsdrehen (vertikale Bearbeitung).

*bearbeitungsverfahren* muß ein gequoteter Ausdruck vom Datentyp Bearbeitungsverfahren sein. Als &rest-Argumente können beliebig viele Paare von gequoteten CONTAX-Variablen und Wertelisten angegeben werden. Diese Angaben schränken die aufgeführten CONTAX-Variablen auf ihre zugehörigen Wertelisten, die Unterbereiche der Datentypen spezifizieren, bereits vor der Propagierung ein.

Bsp.: (werkzeugauswahl '((3 4 5) (9) (12 13)) 'r 'schruppen 'v_Werkzeuglaenge
'(100 125 150 170 180) 'v_Plattendicke '(m_plattendicke))

**prop** [Funktion]

propagiert das aufgebaute Constraint-Netz und liefert bei Erfolg eine Liste der CONTAX-Variablen mit ihren Ausprägungen.

Durch das erste Element der Ergebnisliste wird dabei die Reihenfolge der einzelnen Ausprägungen der CONTAX-Variablen in den folgenden Ergebnistupeln eindeutig festgelegt. Momentan wird bei erfolgreicher globaler Propagierung nur die erste erzeugte Lösung als Ergebnistupel geliefert. Zusätzlich ist zu beachten, daß alle Symbole der Ergebnisliste interne Symbole des Package *werkzeugkonfiguration* sind. Sie müssen demzufolge vom jeweiligen Benutzer durch die Lisp-Funktion *intern* in sein eigenes Package übernommen werden. Hat die Propagierung keinen Erfolg, so wird nil zurückgegeben.

**eingabe** [Funktion]

verwaltet die Dateien Platten.data und Werkzeug.data.

Durch den Aufruf von **eingabe** besteht die Möglichkeit:

- Schneidplatten in die Datei Platten.data zu schreiben oder aus der Datei zu löschen.
- Klemmhalter in die Datei Werkzeug.data zu schreiben oder aus der Datei zu löschen.

Alle Daten, die eingefügt werden sollen, müssen vom verlangten Datentyp sein, der in der Datei constraint.lisp definiert ist. Steht eine benötigte Attributsausprägung nicht zur Verfügung, so muß der betroffene Datentyp entsprechend ergänzt werden. Die Bedienung von **eingabe** soll hier nicht erleutert werden, da die Funktion selbsterklärend ist.

**ausgabe-werkstueck** [Funktion]

gibt die Objekt-Liste des aktuellen Werkstücks auf dem Bildschirm aus.

Die Objekt-Liste enthält zusätzlich zu den Informationen der TEC-REP-Repräsentation für jedes Flächenelement die berechneten Winkel alpha und beta, sowie für Torusabschnittsflächen den Radius der Erzeugenden. Betrachtet man das Schnittbild eines Flächenelementes in der x-z-Ebene, so bezeichnet der Winkel alpha die positive Steigung der Flächenfunktion im 1. Quadranten in Grad und der Winkel beta die negative Steigung in Grad. Für Kreisflächen, Kreisringflächen und Zylindermantelflächen sind demzufolge die Winkel alpha und beta gleich Null. Kegelmantelflächen können, anhängig von der Steigung der Flächenfunktion, nur einen Winkel ungleich Null haben.

**laden-und-pruefen-htable** [Funktion]

lädt die Dateien Platten.data und Werkzeug.data in Hash-Tabellen.

Vor dem Laden findet ein Konsistenztest zwischen den Daten der beiden Dateien und den in constraints.lisp deklarierten Datentypen statt.

**ausgabe-htable** *tabelle* [Funktion]

gibt die durch *tabelle* spezifizierte Hash-Tabelle auf den Bildschirm aus. *tabelle* darf folgende Werte annehmen:

:halter für Werkzeughalter,

:platten für Schneidplatten.

# 5 Implementierung

In diesem Kapitel sollen spezielle Implementierungsfragen erläutert werden und Möglichkeiten, das Werkzeugauswahlmodul an die Wünsche des Benutzers anzupassen. Deshalb ist dieses Kapitel lediglich für solche Benutzergruppen interessant, die Detailkenntnisse vom Programm benötigen, um Änderungen vorzunehmen.

In den folgenden drei Abschnitten werden die grundlegenden Datenstrukturen und implementierungsspezifische Besonderheiten der wichtigsten Programmkomponenten vorgestellt. Der anschließende Abschnitt behandelt das Vorgehen bei konkreten Änderungswünschen.

## 5.1 w_datenbasis.lisp

enthält alle wichtigen Funktionen zur Verwaltung der verfügbaren Drehwerkzeuge, deren Komponenten in den Dateien Platten.data und Werkzeug.data (Datenbasis) abgespeichert sind. Klemmhalter und Schneidplatten lassen sich durch Attributlisten beschreiben (siehe dazu Tabelle 1), deren Datentyp durch die folgenden beiden Definitionen festgelegt wird:

```
(defconstant Werkzeug-Typ '(Werkzeugsystem Spannsystem Plattenform Halterform
                            Halterform Vorschubrichtung Plattengroesse
                            Schafthoehe Schaftbreite Werkzeuglaenge))
(defconstant Platten-Typ  '(Werkzeugsystem Plattenform Freiwinkel Toleranzen
                            Leitstufe&Befestigung Plattengroesse Plattendicke Eckenradius))
```

Die einzelnen Listenelemente von *Werkzeug-Typ* und *Halter-Typ* sind hierarchisch strukturierte Datentypen, die in der Datei constraint.lisp definiert sind. Die Verwaltung der Drehwerkzeuge erfogt in den Hash-Tabellen *Werkzeug-Tabelle* und *Platten-Tabelle*. Interne Funktionen ermöglichen den Datentransfer zwischen der Datenbasis und den Hash-Tabellen in beide Richtungen (lesen und schreiben). Sollen Daten in eine Hash-Tabelle geschrieben werden, so müssen diese Daten vom richtigen Datentyp sein. Die interne Funktion *Typ?* testet zu diesem Zweck, ob der Wert einer Variable ihrem in der Datei constraint.lisp definierten Datentyp entspricht. Eine Hash-Tabelle besteht aus einer Menge von Schlüsseln, denen jeweils eine Werteliste zugeortnet ist. Fügt man einen Schlüssel mit einem Element der Werteliste durch die Funktion *append* zusammen, so ist das Ergebnis eine Attributliste. Dabei ist jedes Element der Werteliste wiederum eine Liste, deren Länge durch die Konstanten *Platten-value-laenge* und *Werkzeug-value-laenge* festgelegt wird. Die Zerlegung der Attributlisten in Schlüssel und Wertelisten ermöglicht sowohl ein komprimiertes Speichern von Attributlisten, als auch einen schnellen Zugriff über Hash-Tabellen.

## 5.2 constraints.lisp

enthält die wesenlichen Komponenten der Wissensbasis des Moduls. Die hierarchisch strukturierten Datentypen der CONTAX-Variablen werden mit der CONTAX-Funktion *def-concept* deklariert. Dabei handelt es sich um taxonomische Hierarchien, die als gerichtete azyklische Graphen repräsentiert werden. Die Elemente der untersten Ebene der hierarchisch strukturierten Datentypen werden als Blattknoten bezeichnet und enthalten die Attributsausprägungen. Superclasses fassen diese zu Mengen gleicher Eigenschaften zusammen. Dieses vereinfacht die Definition von primitiven Constraints und beschleunigt die Propagierung.

Als Constraints werden sowohl primitive Constraints wie auch Prädikat-Constraints verwendet. Bei den meisten primitiven Constraints handelt es sich in Wahrheit um Regeln, weil sie CONTAX-Variablen enthalten, die bereits vor der Propagierung mit Blattknoten von Konzepten instanziert werden, wodurch die Auswertungsrichtung der Constraints festgelegt wird. Durch die Verwendung

| Name | Lisp-Variable | Beschreibung |
|---|---|---|
| test-werkzeughalter | *Werkzeug-Tabelle* | prüft Klemmhalter auf Verfügbarkeit |
| test-schneidplatte | *Platten-Tabelle* | prüft Schneidplatten auf Verfügbarkeit |
| test-winkelbeziehung | *kappa_n* | für die Werkzeugwinkel in der Bezugsebene gilt: $\kappa + \varepsilon + \kappa_n = 180°$, woraus sich die Bedingung Einstellwinkel + Eckenwinkel + *kappa_n* < 180 ergibt |
| test-einstellwinkel | *kappa* | testet, ob Einstellwinkel > *kappa* gilt |
| test-eckenradius | *min_radius* | testet, ob Eckenradius < *min_radius* gilt |

Tabelle 3: Prädikat-Constraints

von Prädikat-Constraints, die durch Lisp-Funktionen definiert werden, besteht die Möglichkeit, auf Lisp eigene Datenstrukturen zuzugreifen. Die in constraint.lisp definierten Prädikat-Constraints, verwenden sowohl die Daten der Hash-Tabellen *Platten-Tabelle* und *Werkzeug-Tabelle* als auch die Variablen *kappa*, *kappa_n* und *min_radius*, die von der Funktion *werkzeugauswahl* instanziert werden (siehe Abschnitt 5.3). *kappa* bezeichnet den minimalen Einstellwinkel des Klemmhalters, *kappa_n* den minimalen Nebenschneiden-Einstellwinkel des Drehwerkzeuges. *min_radius* enthält, falls Torusabschnittsflächen bearbeitet werden sollen, den minimalen Radius der Erzeugenden. Alle definierten Prädikat-Constraints mit ihren verwendeten Lisp-Variablen werden in Tabelle 3 beschrieben.

### 5.3 w_auswahl.lisp

enthält die Benutzerfunktionen zur Ausführung der Werkzeugauswahl, sowie ein Anzahl interner Funktionen zur Auswertung der Eingabedaten. Das Kernstück der Eingabedaten bildet die TEC-REP-Datei, deren Daten von der internen Funktion *read-eingabefile* eingelesen und in eine Liste von CLOS-Objekten überführt werden. Für das Geometrie- und Technologiemakro (GTM) und jedes Flächenelement der TEC-REP-Datei wird ein CLOS-Objekt erzeugt, das entsprechend den Nachbarschaftsbeziehungen in einer linearen Liste eingefügt wird. Die Variable *werkstueck_obj* wird mit dem ersten Element dieser Objekt-Liste der class *werkstueck*, welches die Daten des GTM enthält, instanziert. Die Variable *pathname-werkstueck* enthält den Pfadnamen, unter dem die aktuelle Werkstückrepräsentation zu finden ist. Alle Flächenelemente werden in Objekten der superclass *teil* repräsentiert, die als zusätzliche Informationen die berechneten Winkel alpha und beta (siehe Abschnitt 4.4, ausgabe-werkstueck) enthalten können, sowie die Oberflächengüte und die Härte, falls diese Technologieelemente in der TEC-TEP-Datei enthalten sind. Oberflächengüte und Härte eines Flächelements werden momentan noch nicht als Kriterien für die Werkzeugauswahl herangezogen. Für Torusabschnittsflächen wird zusätzlich noch der Radius der Erzeugenden berechnet.

Die Funktion *werkzeugauswahl* berechnet die Variablen *kappa*, *kappa_n* und *min_radius* (siehe Abschnitt 5.2), setzt CONTAX-Variablen und baut das Constraint-Netz auf. Falls als Bearbeitungsrichtung Längsdrehen gewählt wird, werden die Constraints *test-Winkelbeziehung* und test-Einstellwinkel nicht zum Constraint-Netz hinzugefügt. Werden keine Torusabschnittsflächen bearbeitet, so wird auch das Constraint *test-eckenradius* nicht zum Constraint-Netz hinzugefügt.

Die Funktion *prop* führt die Propagierung des Constraint-Netzes durch. Dabei erfolgt die lokale Propagierung, wie in Abschnitt 3.3 behandelt, mit der minimum-priority „soft". Danach wird, falls eine konsistente Lösung vorliegt, eine globale Propagierung durchgeführt, wobei mit dem keyword :number-of-solutions die Anzahl der Lösungen auf eins gesetzt wird. Bei Erfolg werden anschließend

noch die Ausgabevariablen aus der von CONTAX generierte Lösung erzeugt, wozu die Funktion loesch-elemente verwendet wird. Bei einem Mißerfolg wird nil geliefert.

## 5.4 Änderung der Datenbasis

Falls neue Werkzeuge zur Datenbasis hinzugefügt werden sollen, deren Attributsausprägungen nicht den deklarierten Datentypen entsprechen, ist es nötig, die Datentypen anzupassen. Hierzu müssen lediglich die betroffenen hierarchisch strukturierten Datentypen in der Datei constraint.lisp um die erforderlich Attributausprägungen erweitert werden. Mögliche Erweiterungen der Blattknoten sind bereits als Kommentare angegeben. Werden die Blattknoten der veränderten Datentypen zur Definition primitiver Constraints verwendet, so müssen diese Constraints gegebenenfalls noch angepaßt werden. Nach Durchführung der Änderungen müssen die Dateien constraint.lisp und w_auswahl.lisp in dieser Reihenfolge compiliert und geladen werden. Danach ist es möglich, die Datenbasis unter Verwendung der Funktion *eingabe*, um die neuen Werkzeuge zu erweitern.

# 6 Problemlösungsqualität

Unter Problemlösungsqualität wird i.a. die „objektive" Leistungsfähigkeit eines Programms verstanden, die sich im wesentlichen aus der Ergebnisgüte bei gegebenen Eingabedaten und der benötigten Antwortzeit zusammensetzt. Die Ergebnisgüte muß von einem Experten an Hand von Testfällen beurteilt werden. Dabei ergeben sich offensichtlich Probleme durch die mangelnde Objektivierbarkeit der Beurteilung durch den Experten und die Unvollständigkeit der Testfälle.

In den folgenden zwei Abschnitten wird auf die erreichte Problemlösungsqualität des Werkzeugauswahlmoduls eingegangen. Desweiteren werden Vorschläge zur Verbesserung der Ergebnisgüte und Verringerung der Antwortzeit gemacht.

## 6.1 Ergebnisgüte

Die Ergebnisgüte des Werkzeugauswahlmoduls kann mit „akzeptabel" beurteilt werden, da die konfigurierten Werkzeuge innerhalb der Restriktionen (siehe Abschnitt 2.1) auch unter realen Bedingungen laut Expertenmeinung einsetzbar sind. Allerdings ist das System in den meisten Fällen nicht in der Lage, ein optimales Werkzeug zu konfigurieren, das ein menschlicher Experte bei der gegebenen Problemstellung auswählen würde. Zwar befindet sich das „optimale" Werkzeug nach einer Werkzeugauswahl grundsätzlich in der Lösungsmenge, doch durch die Größe der Lösungsmenge ist das erste, vom System konfigurierte Werkzeug, welches ausgegeben wird, nur selten optimal. Folgende Faktoren führen häufig zu einer zu großen Lösungsmenge:

- Die Gewichtsbehandlung von CONTAX nutzt das Potential der gewichteten Constraints lediglich ansatzweise aus, da sich Constraint-Netze nur stufenweise erweitern lassen.
- Die Gewichtung der einzelnen Constraints ließe sich noch verbessern.
- Bestimmte Aspekte der Werkstückrepräsentation, wie z.B. die Oberflächengüte des Werkstücks, werden bei der Werkzeugauswahl noch nicht berücksichtigt.

Aus diesen Faktoren lassen sich folgende Maßnahmen zur Verbesserung der Ergebnisgüte ableiten:

1. Als einfachste Methode, die bereits mehrfach beim Austesten des Programmes durchgeführt wurde, bietet sich eine Veränderung der Gewichte der Constraints an.
2. Zusätzliche Constraints können die Ergebnisgüte entscheidend verbessern. Dabei können sowohl noch unberücksichtigte Aspekte der Werkzeugauswahl wie die Oberflächengüte und Härte der Flächenelemente zur Definition von Constraints verwendet werden, als auch Bedingungen, die bereits durch definierte Constraints repräsentiert werden, als stärkere Restriktionen in neue Constraints mit niedrigeren Gewichten eingehen. Das zuletztgenannte Vorgehen läßt sich an Hand der Constraints *bestimme-Plattenform* und *bestimme-Plattenform2* erläutern. *bestimme-Plattenform* mit dem Gewicht „medium" beschränkt die Plattenform bei der Bearbeitungsart Schruppen auf dreieckig oder quadratisch. *bestimme-plattenform2* mit dem Gewicht „weak" schränkt die Plattenform bei der Bearbeitungsart Schruppen auf quadratisch ein.
3. Die Erweiterung des CONTAX-Inferenzmechanismus durch Relaxierung würde zu einer, durch die Gewichte der Constraints gelenkten, erheblichen Verkleinerung der Ergebnismenge führen. Dieses ist gleichzusetzen mit einer Verbesserung der Ergebnisgüte, denn die gelieferten Drehwerkzeuge würden entsprechend mehr Constraints erfüllen.

Alle aufgezählten Maßnahmen würden nicht nur das Ergebnis verändern, sondern sich auch auf das Laufzeitverhalten auswirken. Insbesonders die dritte Maßnahme, die die Ergebnisgüte entscheidend verbessern würde, hätte in den meisten Fällen eher negative Effekte auf die Antwortzeit.

## 6.2 Antwortzeit

Die durchschnittliche Antwortzeit des Werkzeugauswahlmoduls läßt sich nur mit dem Begriff „inakzeptabel" klassifizieren, da sie die angestrebte Antwortzeit um das 10-fache überschreitet. Dabei ist das Laufzeitverhalten relativ unabhängig von der Komplexität der dynamischen Eingabedaten, worunter im wesentlichen die Komplexität der Werkstückrepräsentation und die Länge der Bearbeitungsliste (siehe Abschnitt 4.4) zu verstehen ist. Die Werkstückrepräsentation wird in vernachlässigbarer Zeit durch die Funktion *init-werkstück* eingelesen. Mit Hilfe der Bearbeitungsliste werden vor jeder Werkzeugauswahl aus der Werkstückrepräsentation die Winkelrestriktionen berechnet, wodurch die Anzahl der Constraints nicht beeinflußt wird. Durch diese effiziente Vorgehensweise, die die Eigenschaften der zu bearbeitenden Werkstückteile vor der Propagierung in wenigen Variablen zusammenfaßt, wird die Laufzeit der Propagierung unabhängig von der Länge der Bearbeitungsliste.
Die wichtigsten Determinanten des Laufzeitverhaltens sind:

- Der konstante Zeitfaktor für den Aufbau des Constraint-Netzes.
- Die Anzahl der CONTAX-Variablen und die Größe ihrer Wertebereiche.
- Die Anzahl und Komplexität der Constraints.
- Der Inferenzmechanismus.

Wie die durchgeführten Testläufe gezeigt haben, geht die Laufzeit für den Aufbau des Constraint-Netzes mit einem Zeitfaktor von ca. 40% in die momentane Anwendung ein. Durch die Möglichkeit, einmal erzeugte Constraint-Netze abzuspeichern, würde sich dieser Anteil auf nahezu Null reduzieren, falls die Anzahl der durchgeführten Werkzeugkonfigurationen entsprechend groß wäre. Die drei erforderlichen Constraint-Netze würden bereits beim Laden des Moduls erzeugt werden und bei jeder Werkzeugauswahl würde abhängig von den Eingabedaten eines der Constraint-Netze propagiert werden.

Die Anzahl der CONTAX-Variablen und die Größe ihres Wertebereiches hängt im wesentlichen von der Datenbasis ab, da alle vorhandenen Elemente der Datenbasis mit den definierten CONTAX-Variablen beschrieben werden. Zusammen mit der Anzahl der Constraints und der Anzahl der Variablen, über denen die einzelnen Constraints definiert sind, sind diese Faktoren ausschlaggebend für das Laufzeitverhalten der Werkzeugauswahl.

Die Zeitkomplexität eines Constraints hängt exponentiell von der Anzahl der verwendeten Variablen und linear von der Größe ihres jeweiligen Wertebereiches ab. Werden somit Constraints über eine große Anzahl von Variablen definiert, so können sie in der Anwendung bezüglich der Antwortzeit zu einem kritischen Zeitfaktor werden, sofern die Wertebereiche der Variablen nicht schon ausreichend eingeschränkt wurden. Dieses Problem ist signifikant für die Werkzeugauswahl, da die zwei Constraints *test-Werkzeughalter* und *test-Schneidplatte*, die über 9 bzw. 8 Variablen definiert sind, im Constraint-Netz vorhanden sind. Insbesonders das erste der beiden Constraints, das in der momentanen Anwendung ca. 30% der Laufzeit beansprucht, bildet den zeitlichen Engpaß der gesamten Anwendung, da sich aus den 9 Variablen und ihren Wertebereichen rechnerisch über $10^7$ mögliche Kombinationen ergeben.

Infragekommende Maßnahmen zur Verbesserung der Laufzeit wären in diesem Fall einerseits die Reduzierung der Wertebereiche der betroffenen Variablen, womit zwangsläufig eine Verkleinerung der Datenbasis verbunden wäre, andererseits eine gezielte Auswahlstrategie für die Propagierung der aktiven Constraints, die im Inferenzmechanismus zu implementieren wäre. Eine Verringerung der Variablenanzahl des betroffenen Constraints, womit am einfachsten Abhilfe zu schaffen wäre, erscheint bezüglich der Anwendung nicht möglich.

Eine letzte Strategie zur Reduzierung der Antwortzeiten besteht in der Zusammenfassen der Attributslisten der Datenbasis zu höheren Konzepten. Dadurch könnten die auf der Datenbasis operierenden Constraints diese höheren Konzepte bei der Propagierung ausnutzen, anstatt auf den Blattknoten der Konzepte zu arbeiten. In der hier vorgestellten Anwendung scheint dieses Strategie aufgrund der Struktur der Attribute, durch die Werkzeuge und Schneidplatten beschrieben werden, wenig erfogsversprechend zu sein.

Die Auswirkung der Anzahl der Constraints auf das Laufzeitverhalten läßt sich nur schwer abschätzen, da im wesentlichen die „Wirksamkeit" der einzelnen Constraints, die Wertemengen der CONTAX-Variablen möglichst stark einzuschränken, das Laufzeitverhalten beeinflußt. Für die Werkzeugauswahl, die mit maximal 21 Constraints im Constraint-Netz auskommt, spielt die Anzahl der Constraints durch ihre geringe Größenordnung nur eine untergeordnete Rolle und läßt sich, ohne die Ergebnisqualität negativ zu beeinflussen, nicht wesentlich reduzieren. Im Gegensatz dazu erscheint es sogar vorteilhaft, sofern sich der Aufbau des Constraint-Netzes als ein einmaliger Vorgang realisieren läßt, weitere Constraints zu formulieren, die sowohl die Ergebnisqualität der Werkzeugauswahl verbessern, als auch, durch starke Einschränkung der Wertebereiche von Variablen, zu einer Verringerung der Laufzeit des Programmes beitragen.

# Literatur


[1] A. Bernardi, C. Klauck, R. Legleitner: Formalismus zur Repräsentation von Geometrie- und Technologieinformationen als Teil eines Wissensbasierten Produktmodells. Deutsches Forschungzentrum für Künstliche Intelligenz, Document D91-07.

[2] Y. Descotte, J.C. Latombe: Making Compromises among Antagonist Constraints in a Plan. Artificial Intelligence 27, 1985.

[3] R. Legleitner: Entwicklung eines Expertensystems zur Auswahl keramischer Schneidstoffe für die Drehbearbeitung. Universität Kaiserslautern, Lehrstuhl für Fertigungstechnik und Betriebsorganisation, Diplomarbeit 1989.

[4] M. Meyer: Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM environment. Deutsches Forschungszentrum für Künstliche Intelligenz, 1992.

[5] E. Paucksch: Zerspanungstechnik. Vieweg Verlag 1988.

[6] F. Puppe: Einführung in Expertensysteme. Springer Verlag 1988.

[7] SANDVIK GmbH, Geschäftbereich Coromant: Drehwerkzeuge, Außen- und Innenbearbeitung. C-1000:2-GER 1988.

[8] F. Steinle, M. Meyer,C. Jakfeld: CONTAX, Constraint System over Taxonomical Knowledge, User Guide. Deutsches Forschungszentrum für Künstliche Intelligenz, 1992.

# A Domains

Der folgende Auszug aus dem Programmcode des Werkzeugkonfigurationsmuduls enthält die Deklaration der globalen Variablen, die zur Definiton der Prädikat-Constraints benötigt werden, sowie die Deklaration der CONTAX-Datentypen. Die Syntax der CONTAX-Datentypen läßt sich wie folgt beschreiben:

**def-concept** *superconcept ( subconcept$_1$ ... subconcept$_n$ )*
definiert einen abstrakten Datentyp (auch als Klasse oder superconcept bezeichnet), indem alle seine Elemente (subconcepts) aufgezählt werden.

```
;-------------------------------------------------------------------------
;;; Globale Variablen

(defvar *kappa*)            ; minimaler Einstellwinkel
(defvar *kappa_n*)          ; Nebenschneiden-Einstellwinkel
(defvar *min_Radius*)       ; minimaler Radius der zu bearbeitenden
    ; Torusabschnittsflaechen

;-------------------------------------------------------------------------
; Dekleration der domains
; Die Abkuerzungen k, m und g stehen fuer klein, mittel und gross

(def-concept Schneidstoffe (CT515 GC425 GC3015))

(def-concept Werkstueckstoff (Baustahl niedriglegierter_Stahl Gusseisen))
(def-concept Baustahl (Baustahl_niedrig Baustahl_hoch))
(def-concept Baustahl_niedrig (ST33 ST37 ST44 ST50 ST52))
(def-concept Baustahl_hoch (ST60 ST70 ST80))
(def-concept niedriglegierter_Stahl (Einsatzstahl Vergutungsstahl))
(def-concept Einsatzstahl (C15 8MnCr5))
(def-concept Vergutungsstahl (Ck45 Ck65 42CrMo4))
(def-concept Gusseisen (GG GGG))
(def-concept GG (GG20 GG30 GG40))
(def-concept GGG (GGG40 GGG50 GGG60 GGG70 GGG80))

(def-concept Bearbeitungsverfahren (Schruppen Schlichten))

(def-concept Bearbeitungsart (Aussenbearbeitung Innenbearbeitung))

(def-concept Werkstueckstabilitaet (stabil labil))

(def-concept Werkzeugsystem (TMAXP TMAXU))

(def-concept Spannsystem (C bohrung_Spannsystem))
(def-concept bohrung_Spannsystem (P M S))

(def-concept Halterform (B D E F G J K L R S T V))
(def-concept Plattenform (R S T C D))
```

```
(def-concept Achse (z x))
(def-concept Einstellwinkel (m_Einstellwinkel g_Einstellwinkel))
(def-concept m_Einstellwinkel (45 60 72 75))
(def-concept g_Einstellwinkel (90 93 95))

(def-concept Eckenwinkel (0 m_Eckenwinkel g_Eckenwinkel))
(def-concept m_Eckenwinkel (55 60))
(def-concept g_Eckenwinkel (80 90))

(def-concept Vorschubrichtung (N R+L))

(def-concept Schafthoehe (k_Schafthoehe m_Schafthoehe g_Schafthoehe))
(def-concept k_Schafthoehe (10 12))
(def-concept m_Schafthoehe (16 20 25))
(def-concept g_Schafthoehe (32 40))

(def-concept Schaftbreite (k_Schaftbreite m_Schaftbreite g_Schaftbreite))
(def-concept k_Schaftbreite (10 12))
(def-concept m_Schaftbreite (16 20 25))
(def-concept g_Schaftbreite (32 40))

(def-concept Werkzeuglaenge (70 80 100 125 150 170))

(def-concept Plattengroesse (m_Plattengroesse g_Plattengroesse))
(def-concept m_Plattengroesse (11 12))
(def-concept g_Plattengroesse (15 16))


(def-concept Freiwinkel (0 7))

(def-concept Tolleranzen (M))

(def-concept Leitstufe&Befestigung (oben_Befestigung bohrung_Befestigung))
(def-concept oben_Befestigung  (F N R))
(def-concept bohrung_Befestigung (A G M))

(def-concept Plattendicke (k_Plattendicke m_Plattendicke g_Plattendicke))
(def-concept k_Plattendicke (2.38 3.18))
(def-concept m_Plattendicke (3.97 4.76 5.56))
(def-concept g_Plattendicke (6.35))

(def-concept Eckenradius (k_Eckenradius m_Eckenradius g_Eckenradius))
(def-concept k_Eckenradius (0.2 0.4 0.8))
(def-concept m_Eckenradius (1.2))
(def-concept g_Eckenradius (1.6 2.0 2.4))
;-------------------------------------------------------------------------
```

# B Constraints

Der folgende Auszug aus dem Programmcode des Werkzeugkonfigurationsmuduls enthält die Definitionen aller verwendeten CONTAX-Constraints. Dabei haben die Constraints folgende Bedeutung:

**def-primitive-constraint** *constraint-name ( parameter$_1$...parameter$_n$ ) ( domain$_1$...domain$_n$ ) ( value$_{1,1}$...value$_{1,n}$ )...( value$_{m,1}$...value$_{m,n}$ )*
deklariert ein $n$-stelliges primitives Constraint mit dem Namen *constraint-name* und den Parametern *parameter$_1$* bis *parameter$_n$*, die auf ihren jeweiligen Wertebereich durch *domain$_1$* bis *domain$_n$* festgelegt sind, durch Aufzählung aller gültiger Tupel.

**def-lisp-constraint** *constraint-name ( parameter$_1$...parameter$_n$ ) ( domain$_1$...domain$_n$ ) name-of-LISP-function*

deklariert ein $n$-stelliges Prädikat-Constraint mit dem Namen *constraint-name* und den Parametern *parameter$_1$* bis *parameter$_n$*, die auf ihren jeweiligen Wertebereich durch *domain$_1$* bis *domain$_n$* festgelegt sind, durch Bezeichnung einer LISP-Funktion.

Die verwendeten Domains sind im Anhang A aufgeführt.

```
;-------------------------------------------------------------------------
; primitive Constraints, die grundlegende Beziehungen zwischen Variablen
; angeben  (Gewicht: hard)


; gibt an, welchen Eckenwinkel die jeweilige Plattenform hat.
(def-primitive-constraint Eckenwinkel-der-Platten (pf ew)
  (Plattenform Einstellwinkel)
  (R 0) (S 90) (T 60) (C 80) (D 55))


; gibt den Einstellwinkel und die Achse der Halterform an, zu der der
; Einstellwinkel gemessen wird.
(def-primitive-constraint Einstellwinkel-der-Halter (hf a eiw)
  (Halterform Achse Einstellwinkel)
  (B Z 75)
  (D Z 45)
  (E Z 60)
  (F X 90)
  (G Z 90)
  (J Z 93)
  (K X 75)
  (L X 95)
  (L Z 95)
  (R Z 75)
  (S Z 45)
  (T Z 60)
  (V Z 72))

;-------------------------------------------------------------------------
```

```
; allgemeine, primitive Constraints


; Gibt die Eignung des jeweiligen Werkzeugsystems mit seinem zugehoerigen
; Spannsystem fuer die verschiebenen Bearbeitungsarten an.       Gewicht: hard
(def-primitive-constraint Bearbeitung (bv wzs sps)
  (Bearbeitungsverfahren Werkzeugsystem Spannsystem)
 (Schruppen  TMAXP P)
 (Schlichten TMAXU S))


; Gibt die moeglichen Kombinationen von Befestigungsarten an.    Gewicht: hard
(def-primitive-constraint Befestigung (sps lb)
  (Spannsystem Leitstufe&Befestigung)
  (C                    oben_Befestigung)
  (bohrung_Spannsystem bohrung_Befestigung))


; Waehlt den Schneidstoff in Abhaengigkeit von der Bearbeitungsart
; und dem Werkstueckstoff aus.                                   Gewicht: hard
(def-primitive-constraint Schneidstoffwahl (wss bv ss)
  (Werkstueckstoff Bearbeitungsverfahren Schneidstoffe)
  (Baustahl Schlichten CT515)
  (Baustahl Schruppen GC425)
  (niedriglegierter_Stahl Schlichten CT515)
  (niedriglegierter_Stahl Schruppen GC425)
  (Gusseisen Bearbeitungsverfahren GC3015))


; oekonomisches Constraint
; Waehlt eine Leitstufe&Befestigung, die beidseitig
; verwendbar ist.                                                Gewicht: soft
(def-primitive-constraint bestimme-Leitstufe&Befestigung (lb)
  (Leitstufe&Befestigung)
  (A) (F) (G))

;-------------------------------------------------------------------------
; Formulierung von Regeln als primitive Constraints


; Werkstueck stabil und (grosser oder mittlerer Eckenradius)
;                  -> mittlerer Einstellwinkel
; Werkstueck labil -> grosser Einstellwinkel                    Gewicht: medium
(def-primitive-constraint bestimme-Einstellwinkel (wsst er eiw)
  (Werkstueckstabilitaet Eckenradius Einstellwinkel)
  (stabil m_eckenradius m_einstellwinkel)
  (stabil g_eckenradius m_einstellwinkel)
  (stabil k_eckenradius Einstellwinkel)
  (labil  Eckenradius   g_einstellwinkel))
```

```
;Schlichten -> kleiner Eckenwinkel
;Schruppen  -> grosser Eckenwinkel                          Gewicht: hard
(def-primitive-constraint bestimme-Eckenwinkel (bv ew)
  (Bearbeitungsverfahren Eckenwinkel)
  (Schlichten m_Eckenwinkel)
  (Schruppen  g_Eckenwinkel))


; Verschaerfung der vorhergehenden Regel bezueglich der Auswahl des
; kleinstmoeglichen Eckenwinkels.                          Gewicht: medium
(def-primitive-constraint bestimme-Eckenwinkel2 (bv ew)
  (Bearbeitungsverfahren Eckenwinkel)
  (Schlichten 55)
  (Schruppen  g_Eckenwinkel))


; Schruppen  -> grosser oder mittlerer Eckenradius
; Schlichten -> kleiner oder mittlerer Eckenradius           Gewicht: strong
(def-primitive-constraint bestimme-Eckenradius (bv er)
  (Bearbeitungsverfahren Eckenradius)
  (Schruppen  m_Eckenradius)
  (Schruppen  g_Eckenradius)
  (Schlichten k_Eckenradius)
  (Schlichten m_Eckenradius))

; Schruppen  -> grosser Eckenradius
; Schlichten -> kleiner Eckenradius                          Gewicht: weak
(def-primitive-constraint bestimme-Eckenradius2 (bv er)
  (Bearbeitungsverfahren Eckenradius)
  (Schruppen  g_Eckenradius)
  (Schlichten k_Eckenradius))


; Schruppen -> grosse Plattendicke                          Gewicht: strong
(def-primitive-constraint bestimme-Plattendicke (bv pd)
  (Bearbeitungsverfahren Plattendicke)
  (Schruppen g_plattendicke)
  (Schlichten Plattendicke))


; Schruppen -> grosse Schafthoehe und grosse Schaftbreite    Gewicht: weak
(def-primitive-constraint Werkzeugstabilitaet (bv sh sb)
  (bearbeitungsverfahren Schafthoehe Schaftbreite)
  (Schruppen  g_Schafthoehe g_Schaftbreite)
  (Schlichten Schafthoehe   Schaftbreite))


; Schruppen -> grosse Plattengroesse                        Gewicht: weak
```

```
(def-primitive-constraint bestimme-Plattengroesse (bv pg)
  (Bearbeitungsverfahren Plattengroesse)
  (Schruppen  g_Plattengroesse)
  (Schlichten Plattengroesse))

; oekonomisches Constraint
; Schruppen -> bei der Plattenwahl Quadrat oder Dreieck waehlen   Gewicht: medium
(def-primitive-constraint bestimme-Plattenform (bv pf)
  (Bearbeitungsverfahren Plattenform)
  (Schruppen S)
  (Schruppen T)
  (Schlichten Plattenform))


; Verschaerfung der vorhergehenden Regel.
; Schruppen -> bei der Plattenwahl Quadrat waehlen                Gewicht: weak
(def-primitive-constraint bestimme-Plattenform2 (bv pf)
  (Bearbeitungsverfahren Plattenform)
  (Schruppen  S)
  (Schlichten Plattenform))



; weitere Regeln, die noch implementiert werden koennen.
; Schruppen  -> kleiner Freiwinkel   Gewicht: hard
; Schlichten -> grosser Freiwinkel
; Schlichten -> Tolleranz besser als Qualitaet des Werkstuecks   Gewicht : hard

;-----------------------------------------------------------------------
; Praedikat Constraints mit dem Gewicht: hard


; dieses Lisp-Praedikat testet, ob der Einstellwinkel groesser als *kappa*
; ist.

(defun min_einstellwinkel (eiw)
  (> eiw *kappa*))

(def-lisp-constraint test-Einstellwinkel (eiw)
  (Einstellwinkel)
  min_einstellwinkel)


; dieses Lisp-Praedikat testet, ob die Winkelbeziehung in der Bezugsebene
; Einstellwinkel + Eckenwinkel + *kappa_n* < 180 Grad erfuellt ist.

(defun winkelbeziehung (eiw ew)
  (> 180 (+ eiw ew *kappa_n*)))
```

```
(def-lisp-constraint test-Winkelbeziehung (eiw ew)
  (Einstellwinkel Eckenwinkel)
  winkelbeziehung)


; dieses Lisp-Praedikat testet, ob der Eckenradius kleiner als *min_radius*
; ist, der den minimalen Radius der vorhandenen Torusanschnittflaechen angibt.
(defun min_eckenradius (er)
  (< er *min_radius*))

(def-lisp-constraint test-Eckenradius (er)
  (Eckenradius)
  min_eckenradius)


; dieses Lisp-Praedikat testet, ob ein konfigurierter Werkzeughalter im
; hash-table *Werkzeug-Tabelle*, wo die verfuegbaren Werkzeughalter abgelegt
; sind, vorhanden ist.

(defun exist-Werkzeughalter (wzs sps pf hf vr pg sh sb wl)
  (let ((Eintrag (GetEintrag-HTable *Werkzeug-Tabelle*
    (list wzs sps pf hf vr))))
       (when (listp Eintrag)
        (member (list pg sh sb wl) Eintrag :test #'equal))))

(def-lisp-constraint test-Werkzeughalter
  (wzs sps pf hf vr pg sh sb wl)
  (Werkzeugsystem Spannsystem Plattenform Halterform Vorschubrichtung
   Plattengroesse Schafthoehe Schaftbreite Werkzeuglaenge)
  exist-Werkzeughalter)


; dieses Lisp-Praedikat testet, ob eine konfigurierte Schneidplatter im
; hash-table *Platten-Tabelle*, wo die verfuegbaren Schneidplatten abgelegt
; sind, vorhanden ist.

(defun exist-Schneidplatte (wzs pf fw tl lb pg pd er)
  (let ((Eintrag (GetEintrag-HTable *Platten-Tabelle*
    (list wzs pf fw tl lb))))
       (when (listp Eintrag)
(member (list pg pd er) Eintrag :test #'equal))))

(def-lisp-constraint test-Schneidplatte
  (wzs pf fw tl lb pg pd er)
  (Werkzeugsystem Plattenform Freiwinkel Tolleranzen Leitstufe&Befestigung
   Plattengroesse Plattendicke Eckenradius)
  exist-Schneidplatte)
;-----------------------------------------------------------------------
```

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:*
$\Pi_{ODA}$: 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 $\Theta$-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-91-19**
*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-92-14**
*Johannes Schwagereit:* Integration von Graph-Grammatiken und Taxonomien zur Repräsentation von Features in CIM
98 Seiten

**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

**D-92-26**
*Enno Tolzmann:*
Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten