Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Technical Memo
TM-91-12

FEAT-PATR:

Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM

Klaus Becker,
Christoph Klauck,
Johannes Schwagereit

Oktober 1991

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

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

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern und Saarbrücken is a non-profit organization which was founded in 1988 by the shareholder companies ADV/Orga, AEG, IBM, Insiders, Fraunhofer Gesellschaft, GMD, Krupp-Atlas, Mannesmann-Kienzle, Philips, 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

FEAT-PATR:
Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM

Klaus Becker, Christoph Klauck, Johannes Schwagereit

DFKI-TM-91-12

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

FEAT-PATR:
Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM

Klaus **Becker**
Christoph **Klauck**,
Johannes **Schwagereit**

ARC-TEC Projekt
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI GmbH)
Postfach 2080, D-6750 Kaiserslautern, Germany
Telefon: +49631/205-3477
Fax: +49631/205-3210
email: klauck@dfki.uni-kl.de

1. Abstract

In diesem Papier wird aufgezeigt, wie der im Bereich der natürlichsprachlichen Systeme bekannte unifikationsbasierte Grammatikformalismus D-PATR [Kartt86] erweitert werden kann, um im Bereich CAD/CAM zur geometrischen Interpretation von Werkstücken herangezogen werden zu können. Das resultierende System FEAT-PATR demonstriert zum einen die Nützlichkeit einer Analogie zwischen (formalen) Sprachen und der geometrischen Interpretation von Werkstücken und zum anderen die Schwächen bestehender Grammatikformalismen für String-Grammatiken zur geometrischen Interpretation von Werkstücken.

2. Inhalt

3. D-PATR

D-PATR ist eine unifikationsbasierte Entwicklungsumgebung für Grammatiken. Sie eignet sich zur Kodierung einer großen Vielzahl verschiedener Grammatikformalismen, wie etwa die *lexical-functional-grammar* (LFG), *head-driven phrase structure grammar* (HPSG) oder die *functional unification grammar* (FUG). D-PATR bildet aus der eingegebenen Grammatik einen unifikationsbasierten Chart-Parser.

In den nachfolgenden Abschnitten werden die grundlegende Datenstruktur, die Repräsentation einer Grammatik, die Unifikation und die Arbeitsweise in D-PATR vorgestellt.

3.1. DAG's

Die grundlegende Datenstruktur des Formalismus D-PATR stellen gerichtete zusammenhängende azyklische Graphen (Directed Acyclic Graphs (DAG's)) dar. Da der DRS-Konstruktionsalgorithmus teilweise in die Grammatik und damit in diesen Formalismus eingebettet ist, bilden die DAG's auch hier die grundlegende Datenstruktur. In diesem Abschnitt werden die benötigten Definitionen und das Wesen dieser Graphen dargestellt.

Intuitiv betrachtet, ist ein DAG für den Formalismus ein Informationen beinhaltendes Objekt, daß ein anderes Objekt durch die Spezifizierung von Werten für verschiedene Attribute des Objektes beschreibt oder repräsentiert. DAG's stellen partielle Informationen des beschriebenen Objektes zur Verfügung.

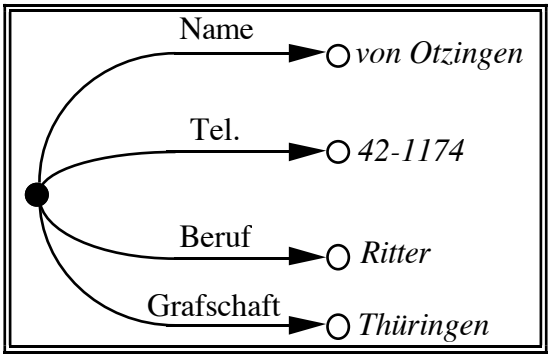
Mathematisch betrachtet werden DAG's durch folgende Definitionen beschrieben.

1. Ein *gerichteter Graph* Φ besteht aus einer Menge $V = V(\Phi)$, den *Knoten* von Φ, und einer Menge $E = E(\Phi)$ von geordneten Paaren $\langle \xi,\upsilon \rangle$, mit $\xi,\upsilon \in V$, den *Kanten* von Φ.
2. Zwei Knoten $\xi,\upsilon \in V$ heißen *direkt verbunden*, wenn $\langle \xi,\upsilon \rangle \in E$ oder $\langle \upsilon,\xi \rangle \in E$ gilt.
3. Ein gerichteter Graph Φ heißt *zusammenhängend*, wenn es für je zwei Knoten $\xi,\upsilon \in V$ eine Menge $\{\xi_1, \xi_2, ..., \xi_n\} \subseteq V$ gibt, wobei ξ_i mit ξ_{i+1} direkt verbunden ist ($i \square \in \square \{0, ..., n-1\}$), die ξ und υ verbindet, das heißt für die gilt, daß ξ und ξ_0 und υ und ξ_n direkt verbunden sind.
4. Ein gerichteter Graph Φ heißt *azyklisch*, wenn er keine Kantenfolge der Form $\langle \square\upsilon\square,\square\xi_0,\square\ldots,\square\xi_n\ ,\ \upsilon \rangle$ enthält.

Die Standardnotation für DAG's sind Matrizen von Attribut-Werte Paaren oder gezeichnete Graphen, wie im nachfolgenden Beispiel demonstriert. Die Attribute sind markierte Kanten und die Werte sind Knoten in einem DAG.

Name	*von Otzingen*
Tel.	*42-1174*
Beruf	*Ritter*
Grafschaft	*Thüringen*

Attribute sind hier *Name*, *Tel.*, *Beruf* und *Grafschaft*, und die zugehörigen Werte sind *von Otzingen*, *42-1174*, *Ritter* und *Thüringen*. In dem gezeichneten Graph werden die Attribute als markierte Kanten und deren Werte als Knoten repräsentiert.



Innerhalb der DAG's ist eine Schachtelung möglich, das heißt ein Wert eines Attributes in einem DAG kann wieder ein DAG sein.

Eine aus der Rekursion der DAG's resultierende Notation sind *Pfade*. Ein Pfad ist eine (endliche) Sequenz von Attributen und beschreibt einen (partiellen) Weg in einem DAG. Atomare Werte sind DAG's mit dem leeren Pfad als einzigen Pfad.

Eine wichtige Eigenschaft der DAG's ist die *Subsumption*; das heißt ein DAG kann mehr Informationen beinhalten als ein anderer. Daraus resultierend läßt sich folgende reflexive Partialordnung $\preceq$ definieren.

Es gilt $dag_1 \leq dag_2$, falls

1. dag_1 und dag_2 sind atomar und $dag_1 = dag_2$, oder
2. $\forall$ Attribute l, mit $|l,n_1| \in dag_1$, existiert ein Attribut k, mit $|k,n_2| \in dag_2$ und $k = l$ und $n_1 \leq n_2$ und $\forall$ Attribute $l_1,l_2 \in dag_1$ mit structure-sharing, haben auch die zugehörigen Attribute $k_1, k_2 \in dag_2$ structure-sharing. Die Notation $|l,n|$ steht für das Attribut l mit Wert n.

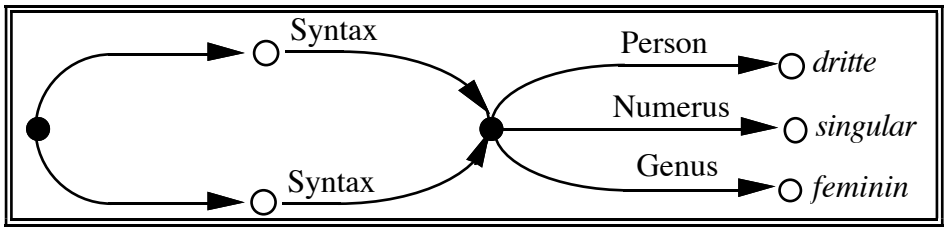
Der leere DAG ist das Top-Element ⊤ und der inkonsistente DAG, das heißt der DAG mit allen Informationen, ist das Bottom-Element ⊥.

3.2. Das structure-sharing Prinzip

Das Prinzip des *structure-sharing* wird in Bezug auf die von der Grammatik und der DRS eines Satzes verwendeten Datenstruktur der DAG's aufgezeigt.

Im wesentlichen ermöglicht das Prinzip, das zwei oder mehrere disjunkte Attribute oder Pfade in einem DAG ihre Werte durch ein und denselben DAG spezifizieren können. Dazu ist nachfolgend ein Beispiel illustriert. Zur Darstellung des structure-sharing werden in < und > eingeklammerte Pfade verwendet, die in der Darstellung der DAG's als Pointer realisiert sind.

Subjekt	Syntax	Person	*dritte*
		Numerus	*singular*
		Genus	*feminin*
Prädikat	Syntax	<Subjekt Syntax>	



Es ist von Bedeutung, den Unterschied zwischen structure-sharing und dem Fall strukturell gleicher Werte zu erkennen. In dem nächsten Beispiel sind strukturell gleiche Werte vorhanden.

Manager	Name	*Rössner*	
	VersNr	*42-1174-R-88*	
	Sekretär	Name	*Müller*
Sub-Manager	Name	*Lüdenscheidt*	
	VersNr	*13-2712-L-63*	
	Sekretär	Name	*Müller*

Eine Personalnummer oder ähnliches könnte darüber Aufschluß geben, daß es sich hier um zwei verschiedene Sekretäre handelt. Sie könnte aber natürlich auch darüber Aufschluß geben, daß es sich um ein und denselben Sekretär handelt.

Nachfolgend ist ein weiteres Beispiel einer Struktur mit structure-sharing gegeben. Hier wird durch dieses Prinzip eindeutig ausgesagt, daß es sich um ein und denselben Sekretär handelt.

Manager	Name	*Rössner*	
	VersNr	*42-1174-R-88*	
	Sekretär	Name	*Müller*
Sub-Manager	Name	*Lüdenscheidt*	
	VersNr	*13-2712-L-63*	
	Sekretär	<Manager	Sekretär>

Von Vorteil ist dieses Prinzip vor allem im Zusammenhang mit den Diskursreferenten. Jeder Diskursreferent, beziehungsweise sein DAG, wird nur einmal explizit aufgeführt; zu allen anderen Stellen besteht eine Verbindung über das structure-sharing Prinzip, also im Endeffekt über Pointer.

3.3. Der Formalismus

Eine *Regel* in D-PATR ist eine Liste von Konstituenten (*constituent labels*), denen Spezifikationen (*specifications*) folgen können. Spezifikationen stellen Bedingungen über eine oder mehrere Konstituenten der Regel dar. Um diese Referenz zu beschreiben, erhält jede Konstituente eine Nummer, wobei die Liste der Konstituenten, von links, mit Null beginnend, nach rechts durchnummeriert wird. Die einfache Regel S → NP VP erhält zum Beispiel in D-PATR die Gestalt (S NP VP). Mittels 0 kann auf S referiert werden und mittels 2 auf VP.

Eine *Spezifikation* wird als zweielementige Liste mit folgender Form dargestellt :

({ Attribut | Pfad } { Pfad | Wert })

Attribut ist hierbei ein Atom, *Pfad* eine Liste von Atomen und *Wert* ist entweder ein Atom oder eine Liste von Spezifikationen.

Sei n, m $\in N$, dann ist

$$
\begin{array}{c}
(x_0\ x_1\ \ldots\ x_n \\
spec_1 \\
spec_2 \\
\cdots \\
spec_n\ )
\end{array}
$$

eine Regel, wobei$spec_i$ (i = 1 ... m) eine Spezifikation der folgenden Art ist :

1. ((k $attribut_k$) $wert_k$),
2. ((k $pfad_k$) $wert_k$),
3. ((k $attribut_k$) (l $pfad_l$)) oder
4. ((k $pfad_k$) (l $pfad_l$))

wo k, l $\in N$,1 $\leq$ k, l $\leq$ n; k und l referieren auf die Konstituenten x_k und x_l. $pfad_i$ ist eine Liste der Attribute a_1, a_2, ..., $a_{s(i)}$, mit s(i) = Länge des Pfades $pfad_i$.

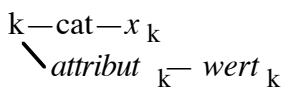
Als Repräsentation solcher Regeln in D-PATR dienen DAG's. In diesen Graphen werden die Konstituenten der Regel als markierte Kanten (Attribut oder auch *feature* genannt) und als Knoten (Wert genannt) dargestellt. Die Kategorien der Kanten werden in einem reservierten Attribut *cat* dargestellt.

Die Repräsentation der Liste der Konstituenten x_0 x_1 ... x_n als DAG sieht wie folgt aus:

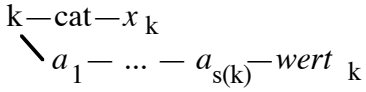
$$0 — \text{cat} — x_0$$
$$1 — \text{cat} — x_1$$
$$\cdots$$
$$n — \text{cat} — x_n$$

Welche Darstellung erhalten die Spezifikationen ?

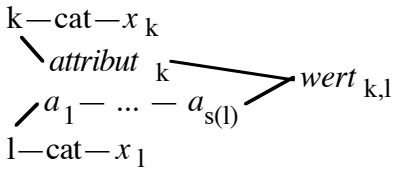
Spezifikationen der ersten Art werden repräsentiert als :



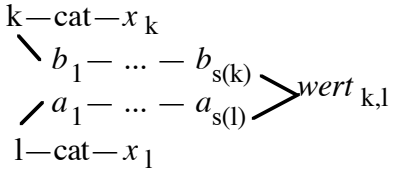
Spezifikationen der zweiten Art werden repräsentiert als :



Spezifikationen der dritten Art werden repräsentiert als :



Spezifikationen der vierten Art werden repräsentiert als :



Zu der Darstellung der Spezifikationen der dritten und vierten Art ist zu bemerken, daß der Wert $wert_{k,l}$ jeweils nur einmal explizit notiert wird; an den anderen Stellen steht ein entsprechender Pointer auf diesen Wert. Dies resultiert daraus, daß die Spezifikationen die Werte des Attributes und des Pfades, beziehungsweise der beiden Pfade gleichsetzt und D-PATR zur Realisierung dieser Gleichsetzung das structure-sharing Prinzip verwendet.

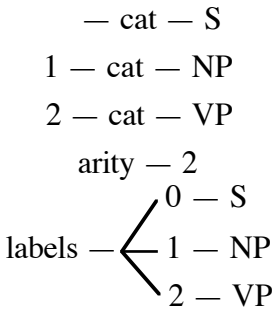
Als äquivalente Repräsentation zu der Graphendarstellung werden von der Benutzeroberfläche Matrizen verwendet.

Die drei möglichen äquivalenten visuellen Repräsentationsarten werden nun an dem Beispiel der Regel S → NP VP veranschaulicht.

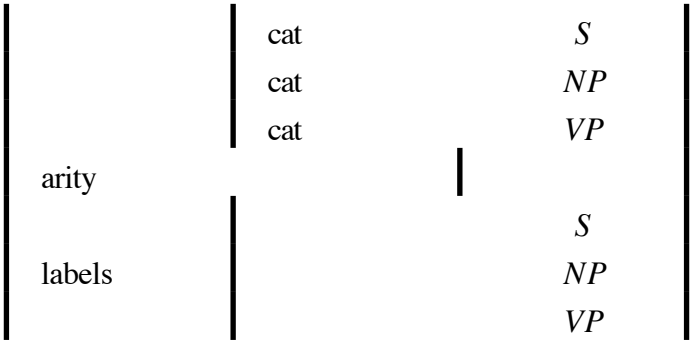
1. Als Notation in der Grammatik:

(S NP VP)

2. Als DAG in D-PATR :



3. Als Matrize in der Benutzeroberfläche :



3.4. Unifikation

Ein wesentlicher Punkt in D-PATR spielt die Unifikation. Eine Spezifikation (*linke-Seite rechte-Seite*) ist eigentlich nur eine andere Schreibweise für unifiziere(*linke-Seite rechte-Seite*).

Die Unifikation in D-PATR ist eine spezielle Art rekursiver Mengenvereinigung, die auf Attribut-Werte Paare definiert ist.

Das Ergebnis einer Unifikation zweier DAG's dag_1 und dag_2 ist der DAG *dag*, wobei

1. $dag = dag_1$, falls $dag_1 = dag_2$,
2. $dag = dag_1$, falls dag_1 atomar und dag_2 leer ist,
3. $dag = dag_2$, falls dag_2 atomar und dag_1 leer ist,
4. wenn weder dag_1 noch dag_2 atomar sind, dann
   $\forall$ Attribute l, mit $|l,n_1| \in dag_1$ und $|l,n_2| \in dag_2$, gilt $|l,$ *unifiziere(* n_1, n_2 *)*$| \in dag$ und
   $\forall$ Attribute l, mit $|l,n| \in \{ n_1 \cap n_2 \} \setminus \{ n_1 \cup n_2 \}$, gilt $|l, n| \in dag$.
   Die Notation $|l, n|$ steht für das Attribut l mit dem nachfolgenden Pfad und seinem Wert n oder nur seinen Wert n.
5. In allen anderen Fällen schlägt die Unifikation fehl.

Das Fehlschlagen der Unifikation wirkt sich dadurch aus, daß der vollständige DAG gestrichen wird, unabhängig von der Situation der Satzanalyse. Die Unifikation arbeitet destruktiv (D-PATR bietet auch die Möglichkeit, die Unifikation nicht destruktiv durchzuführen, das heißt die Strukturen der zu unifizierenden DAG's bleiben erhalten.), das heißt nach der Unifikation der DAG's dag_1 und dag_2 gilt $dag_1 = dag = dag_2$.

In Fall 2. und 3. wird die Unifikation zum Propagieren von Information verwendet, wogegen in Fall 1. existierende Informationen überprüft werden. Fall 4. stellt die Rekursion der Definition dar.

Am Rande seien folgende interessante Eigenschaften der Unifikation notiert.

1. **Idempotenz** :
   unifiziere(dag_1, dag_1) = dag_1
2. **Kommutativität** :
   unifiziere(dag_1, dag_2) = unifiziere(dag_2, dag_1)
3. **Assoziativität** :
   unifiziere(dag_1, unifiziere(dag_2, dag_3)) = unifiziere(unifiziere(dag_1, dag_2), dag_3)

4. Neutrales Element :

unifiziere(ε, *dag*) = unifiziere (*dag*, ε) = *dag*, wo ε der leere DAG ist.

Man hat also einen kommutativen Monoiden *(Menge aller DAG's, unifiziere)* mit Idempotenz.

Auf die verschiedenen Arten der Spezifikationen wirkt sich diese Definition der Unifikation wie folgt aus.

- Erste und zweite Art :
  - falls kein Wert für das Attribut $attribut_k$ oder den Pfad $pfad_k$ existiert, führt die Spezifikation als Wert für das Attribut oder den Pfad $wert_k$ in den DAG als neue statische Information ein.
  - falls ein Wert *wert* für das Attribut oder den Pfad existiert, werden die Werte *wert* und $wert_k$ gemäß der Definition der Unifikation unifiziert.

Mit dieser Konstruktion kann Information, die zur Zeit der Konstruktion der Grammatik bekannt ist, in den DAG integriert und überprüft werden.

- Dritte und vierte Art :
  - falls ein Pfad $pfad_i$ oder das Attribut $attribut_k$ ein Wert hat und der andere Pfad $pfad_j$ oder das Attribut $attribut_k$ kein Wert hat, kann neue, dynamisch berechnete Information von einer Konstituenten zu einer anderen propagiert werden.
  - falls beide Pfade oder das Attribut und der Pfad einen Wert haben, kann die Kompatibilität zweier Konstituenten überprüft werden.

Mit dieser Konstruktion kann die Vererbung von Informationen realisiert werden.

Als einfaches Beispiel soll hier die klassische Phrasenstruktur-Regel S → NP VP betrachtet werden. Die Notation in der Grammatik sei wie folgt :

```
(S  NP VP
        (0              (1))
        (0              (2))
        ((1 number)     (2 number))
        ((1 person)     (2 person)))
```

Mit den ersten beiden Spezifikationen wird eine Vererbung aller Informationen der Töchter NP und VP an die Mutter S bewirkt; die letzten beiden bewirken eine Kompatibilitätsprüfung der Konstituenten NP und VP.

Um also eine Überprüfung der Kompatibilität oder eine Vererbung durchführen zu können, muß man wissen, an welchen Punkten Informationen bei den Attributen oder Pfaden vorhanden sind oder noch nicht bekannt sind.

3.5. Die Funktionsbeschreibungen

Zunächst werden die das Gerüst des Chart-Parsers bildenden Funktionen *chart-parse, predict, try-to-extend, add-inactive-edge, add-active-edge* beschrieben. Anschließend erfolgt eine Erläuterung der für die Unifikation verantwortlichen Funktionen *match-edges, copy-unify, unify, unify-complex-macro, unify-atomic-macro.*

Es ist zu beachten, daß nur die für das prinzipielle Verständnis der Vorgehensweise des D-PATR benötigten Funktionsaufrufe beschrieben werden.

Funktion: chart-parse
aufgerufen in: -
Aufrufe: u.a. add-word-to-chart
Eingabe: Der zu parsende String
Ausgabe: Das Ergebnis des Parsers oder eine Fehlermeldung

Für jedes Wort des Eingabestrings wird die Funktion *add-word-to-chart* aufgerufen. Als Parameter werden das Wort, der linke und ein neu kreierter rechter Knoten mitgegeben.

Funktion: add-word-to-chart
aufgerufen in: chart-parse
Aufrufe: add-inactive-edge
Eingabe: Wort, Start- und Endknoten

Für jede dem dag-cache entnommene Bedeutung des Wortes wird *add-inactive-edge* mit Parametern Start-, Endknoten und Wortbedeutung aufgerufen.

Funktion: add-inactive-edge
aufgerufen in: try-to-extend, add-word-to-chart
Aufrufe: try-to-extend, get-matching-non-terms

Eingabe: Start- und Endknoten der neuen inaktiven Kante, instantiierte Regel, Originalregel, Kanten, die zum Startknoten führen, Quelle, optional: Verbindungen
Ausgabe: -

Es wird eine neue inaktive Kante zwischen Start- und Endknoten kreiert. Fehlt die Eingabe der Verbindungen, so wird für jede aktive Kante, die unter dem slot *active-in* vermerkt ist, getestet, ob die zu expandierende Stelle der zugehörigen Regel (*needed*) mit dem Kopf der instantiierten Regel unifizierbar ist. Ist dies der Fall, so wird *try-to-extend* mit den Parametern aktive Kante, inaktive Kante aufgerufen.

Wurden die Verbindungen der Eingabe beigefügt, so wird für jede Verbindungskante *try-to-extend* mit den Parametern Verbindungskante und neue inaktive Kante aufgerufen.

Funktion: try-to-extend
aufgerufen in: u.a. add-inactive-edge
Aufrufe: add-inactive-edge, add-active-edge, match-edges
Eingabe: .a. aktive Kante, inaktive Kante
Ausgabe: -

Es wird getestet, ob der Kopf der instantiierten Regel der inaktiven Kante mit der zu expandierenden Stelle der instantiierten Regel der aktiven Kante (*needed*) unifiziert. Das Ergebnis der Unifikation wird in der Variablen *new-feats* abgelegt. Konnte erfolgreich unifiziert werden, so wird zwischen zwei Fällen unterschieden.

1. Fall: *needed* beendet die Regel (d.h. *needed* = Stelligkeit der Regel der aktiven Kante (*arity*)).
Es erfolgt ein Aufruf von *add-inactive-edge* mit u.a. folgenden Parametern: Startknoten der aktiven Kante, Endknoten der inaktiven Kante, *new-feats* . Dies bedeutet, daß hier bottom-up geparst wird.

2. Fall: *needed* < *arity* . *needed* wird um 1 inkrementiert, d.h. der nächste Teil der rechten Seite der Regel muß in Zukunft expandiert werden.
Es wird eine neue Kante kreiert, deren Startknoten der Startknoten der aktiven Kante ist (*left-vertex*), und deren Zielknoten der Zielknoten der inaktiven Kante ist (*right-vertex*).

Schließlich erfolgt ein Aufruf von *add-active-edge* mit u.a. folgenden Parametern: *left-vertex* , *right-vertex* , neue Kante, *new-feats* .

Funktion:	add-active-edge
aufgerufen in:	predict
Aufrufe:	u.a. predict
Eingabe:	Start- und Endknoten, Kante, instantiierte und uninstantiierte Regel, Stelle, an der die Regel zu expandieren ist (*needed*)
Ausgabe:	-

Die eingegebene Kante wird als neue aktive Kante zwischen Start- und Endknoten eingefügt.

Um die durch *needed* indizierte Stelle der zur Kante gehörenden Regel zu expandieren (Start eines top-down parse), wird die Funktion *predict* mit folgenden Parametern aufgerufen: durch *needed* indizierte Regelkomponente der instantiierten Regel, durch *needed* indizierte Regelkomponente der uninstantiierten Regel, Zielknoten der neuen aktiven Kante, neue aktive Kante.

Funktion:	predict
aufgerufen in:	add-active-edge, chart-parse-pre-processing-for-grammar
Aufrufe:	u.a. add-active-edge, compatible-p, get-matching-rules
Eingabe:	instantiierter Regelkopf, uninstantiierter Regelkopf, Startknoten des top-down-parse (*vertex*), optional: Kante, die zuletzt eingefügt wurde und deren Zielknoten *vertex* ist (*from-edge*).
Ausgabe:	-

predict ist für den top-down-parse-Teil des D-PATR zuständig.

Zunächst werden die Regeln zusammengestellt, deren Kopf mit dem instantiierten Regelkopf der Eingabe unifizieren.

Für jede dieser Regeln geschieht folgendes:

1.Fall: Die Regel ist schon mit einer Kante assoziiert im active-rules-slot des *vertex* vermerkt.
In diesem Fall wird die *from-edge* zu der im connections-slot jener assoziierten Kante stehenden Liste hinzugefügt.

2. Fall: Die Regel ist noch nicht mit einer Kante assoziiert im active-rules-slot des *vertex* vermerkt.
Es wird eine neue Kante erzeugt, deren needed-slot mit 1 initialisiert wird (d.h. als nächstes muß die erste Komponente der rechten Seite der Regel expandiert werden). Start- und Zielknoten dieser Kante ist *vertex* . Zusätzlich wird die Regel mit der neuen Kante assoziiert und in die Liste des active-rules-slot des *vertex* eingefügt. Danach wird *add-active-edge* mit den Parametern *vertex*,*vertex*, neue Kante, Regel (Bedeutung: instantiierte Regel), Regel (Bedeutung: uninstantiierte Regel), 1 (Bedeutung: *needed*) aufgerufen. Von dort wird wieder rekursiv *predict* mit der durch *needed* indizierten Komponente der Regel als neuem zu expandierenden Regelkopf aufgerufen.

Funktion: match-edges
aufgerufen in: try-to-extend
Aufrufe: u.a. copy-unify
Eingabe: u.a. aktive und inaktive Kante
Ausgabe: Ergebnis der Unifikation

Aufruf der funktion *copy-unify* mit folgenden Parametern: Kopf der instantiierten Regel der inaktiven Kante, instantiierte Regel der aktiven Kante.

Funktion: copy-unify
aufgerufen in: match-edges
Aufrufe: u.a. unify
Eingabe: dag1, dag2
Ausgabe: Ergebnis der Unifikation

U.a. erfolgt ein Aufruf der Funktion *unify* mit u.a. den Parametern *dag1* und*dag2* .

Funktion: unify
aufgerufen in: u.a. copy-unify, compatible-p
Aufrufe: u.a unify-atomic-macro, unify-complex-macro
Eingabe: u.a. dag1, dag2
Ausgabe: Ergebnis der Unifikation

Sind die dags syntaktisch gleich, oder ist einer der dags eine leere Liste, so wird der dag ausgegeben, der keine leere Liste ist. Ist einer der dags atomar, so erfolgt ein Aufruf von *unify-atomic-macro*. Sonst erfolgt ein Aufruf von *unify-complex-macro*.

Funktion: unify-complex-macro
aufgerufen in: unify
Aufrufe: u.a. unify
Eingabe: complex-dag1, complex-dag2
Ausgabe: Ergebnis der Unifikation der beiden komplexen dags

Für jeden arc des *complex-dag2* wird getestet, ob ein arc gleichen Namens im *complex-dag1* existiert. Wenn nicht, dann wird der arc an den dagbody des complex-dag1 angefügt. Wenn doch, dann wird u.a. die Funktion *unify* mit folgenden Parametern aufgerufen: Wert des arc des *complex-dag1*, Wert des entsprechenden arc des *complex-dag2* .

Funktion: unify-atomic-macro
aufgerufen in: unify
Aufrufe: -
Eingabe: dag1, dag2
Ausgabe: Ergebnis der Unifikation

Die beiden dags unifizieren, falls die beiden dag-bodies syntaktisch gleich sind, oder falls einer der dagbodies das Symbol *any* ist und der andere nicht das Symbol *none* .

3.6. Ein Beispiellauf des D-PATR

Um die Arbeitsweise des D-PATR noch besser verständlich darzustellen, soll anhand eines kleinen Beispiels die Vorgehensweise des Systems erläutert werden.

Gegeben seien folgende Regeln (bzw. Lexikoneinträge):

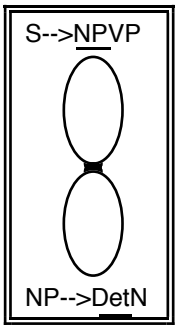
S --> NP VP
NP --> Det N
VP --> Aux V
Det --> the
N --> monkey
Aux --> has
V --> slept

Die 4 Lexikoneinträge werden im folgenden auch Regeln genannt. Man muß sich aber vergegenwärtigen, daß sie nicht im *rule-dag* stehen, sondern nur im *dag-cache*.

Der folgende Satz soll geparst werden:

The monkey has slept.

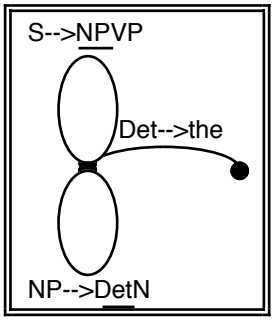
Zunächst wird vom Grammatik-file aus die Funktion *set-top-category* aufgerufen. Von dieser Funktion erfolgt ein Aufruf von *chart-parse-pre-processing-for-grammar*. Diese Funktion wiederum ruft *predict* mit dem Argument *top-cat-dag* (Wert: S) auf. *Predict* führt mit dem eingegebenen Regelkopf im Wechsel mit *add-active-edge* einen top-down-depth-first-parse durch. Es entsteht folgende Struktur:



Innerhalb einer Regel ist die Regelkomponente unterstrichen, die als nächste zu expandieren ist.

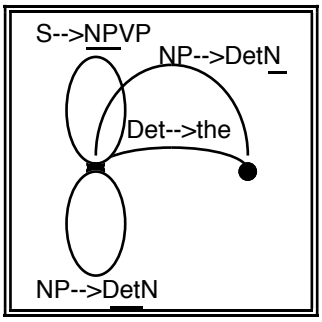
Kanten, deren Regeln eine unterstrichene Komponente enthalten, sind aktive Kanten, d.h. die unterstrichene Komponente der Regel und die Komponenten rechts von ihr müssen noch expandiert werden.

Der Benutzer ruft jetzt die Funktion *chart-parse* mit dem zu parsenden String als Argument auf. Für jedes Wort des String wird die Funktion *add-word-to-chart* aktiviert. Diese wiederum ruft die Funktion *add-inactive-edge* auf, die eine neue inaktive Kante mit dem Eintrag Det-->the erzeugt und in den chart einfügt:

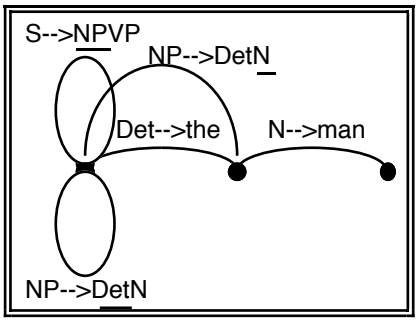


add-inactive-edge aktiviert die Funktion *try-to-extend* mit u.a. folgenden Argumenten: aktive Kante mit Regel-slot NP-->DetN, inaktive Kante mit Regel-slot Det-->the.

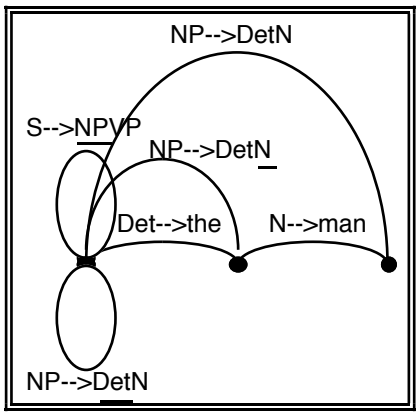
Da die Regel der aktiven Kante noch nicht fertig abgearbeitet ist, inkrementiert *try-to-extend* das needed-flag der aktiven Kante um 1, kreiert aus den beiden Kanten eine neue Kante und ruft *add-active-edge* mit dieser Kante als Argument auf. Es entsteht folgende Struktur:



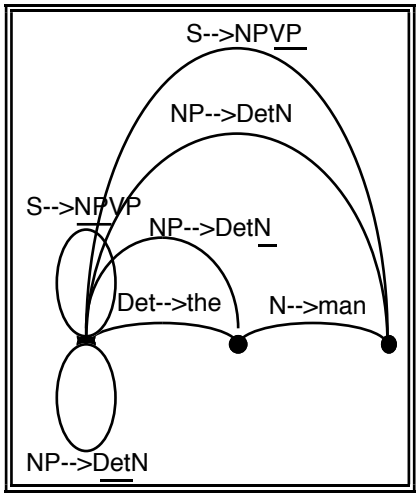
add-active-edge ruft nun *predict* auf, um das N der Regel der neuen Kante zu expandieren. Da N kein Kopf einer Regel ist, beendet sich *predict* ohne weitere Funktionsaufrufe. Auch andere aufrufende Funktionen beenden sich, und es erfolgt ein Rücksprung zu *chart-parse*. Nun wird wiederum *add-word-to-chart* mit dem nächsten Wort des Eingabestrings aktiviert. Nach erneutem Aufruf von *add-inactive-edge* ergibt sich folgendes Bild:



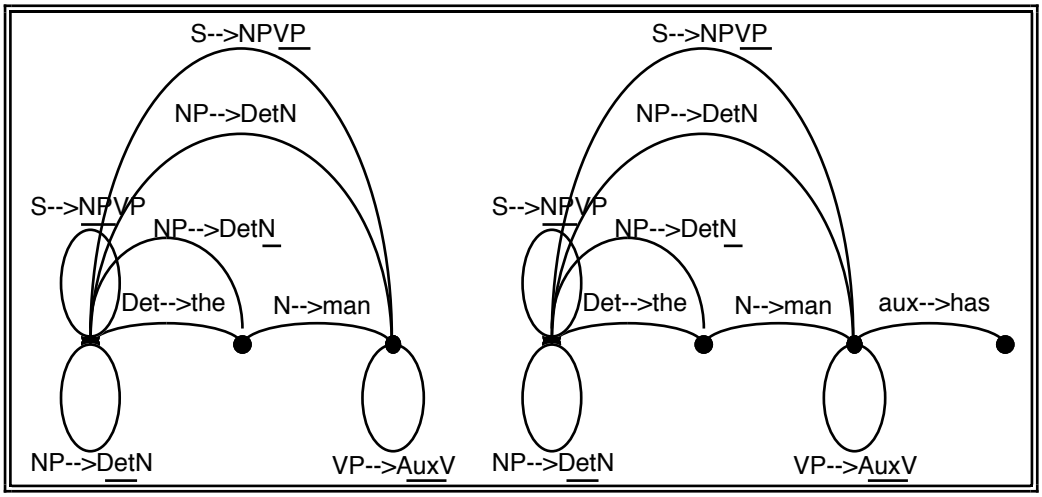
add-inactive-edge ruft *try-to-extend* mit den Kanten der Regeln NP-->DetN und N-->man auf. Von dort aus wird wiederum *add-inactive-edge* aktiviert und es entsteht folgende Struktur:

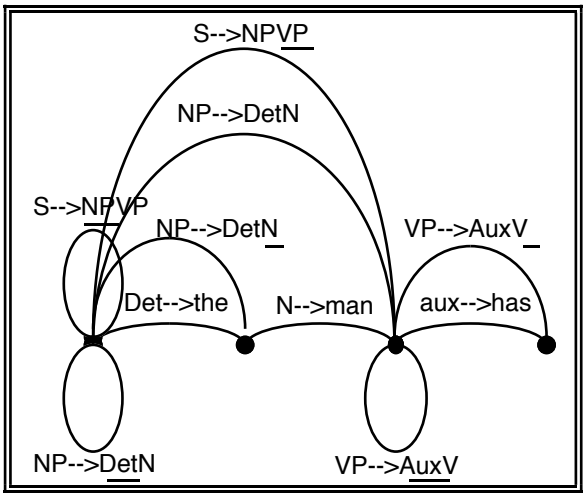


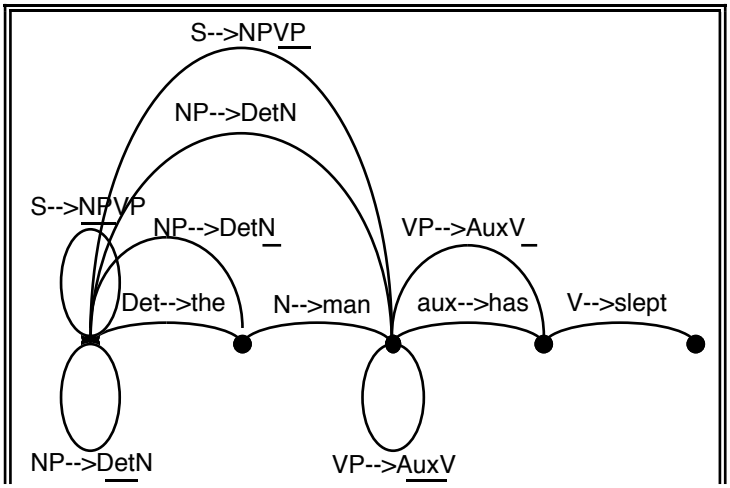
Es folgt ein Aufruf von *try-to-extend* mit den Kanten der Regeln S-->NPVP und NP-->DetN. Nach der folgenden Aktivierung von *add-active-edge* ergibt sich folgendes Bild:

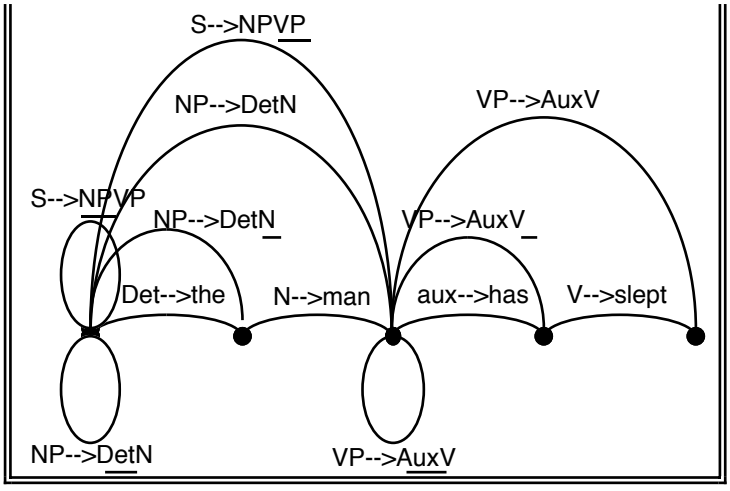


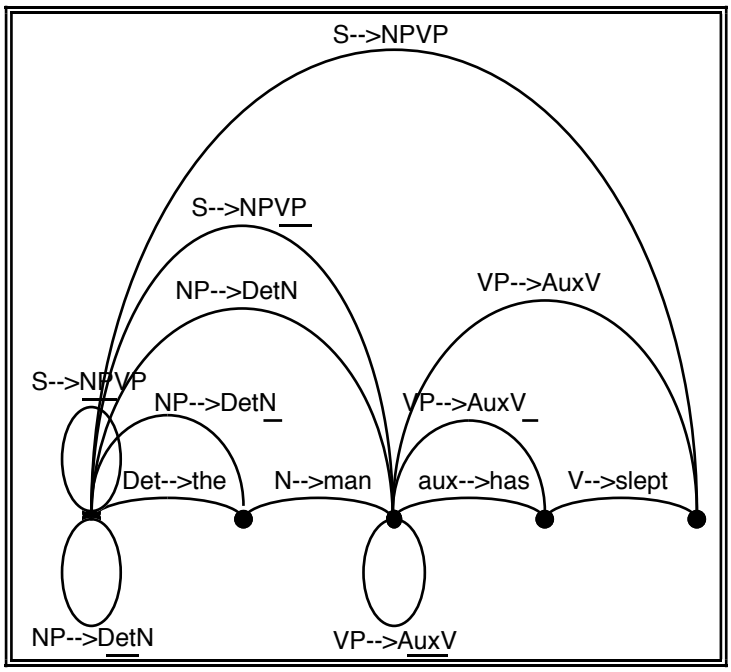
Nun erfolgt über *predict* eine Expandierung der VP-Komponente der Regel der neuen aktiven Kante. Die folgenden Schritte laufen analog zu den vorherigen ab und werden deshalb nur durch Graphiken veranschaulicht.




S-->NPVP
NP-->DetN
S-->NPVP
NP-->DetN
VP-->AuxV
Det-->the
N-->man
aux-->has
NP-->DetN
VP-->AuxV


S-->NPVP
NP-->DetN
S-->NPVP
NP-->DetN
VP-->AuxV
Det-->the
N-->man
aux-->has
V-->slept
NP-->DetN
VP-->AuxV


S-->NPVP
NP-->DetN
VP-->AuxV
S-->NPVP
NP-->DetN
VP-->AuxV
Det-->the
N-->man
aux-->has
V-->slept
NP-->DetN
VP-->AuxV

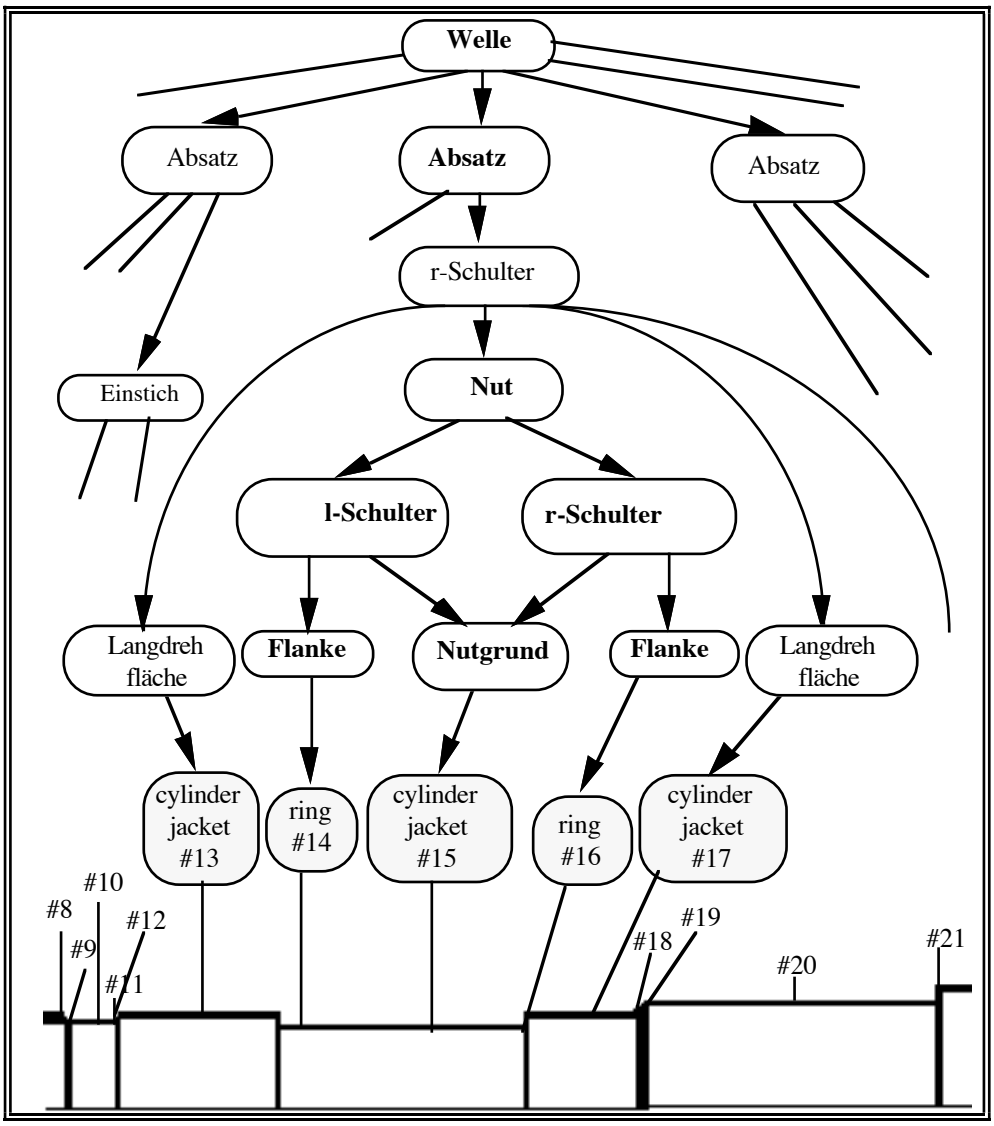

S-->NPVP
S-->NPVP
NP-->DetN
VP-->AuxV
S-->NPVP
NP-->DetN
VP-->AuxV
Det-->the
N-->man
aux-->has
V-->slept
NP-->DetN
VP-->AuxV

4. Features und Feature-Recognition

Die Werkstückbeschreibung eines CAD-Modells ist meist in Form einer elementaren Geometrie und Topologie gehalten, die für eine direkte Verarbeitung durch CAPP-Systeme (Computer Aided Process Planning) ungeeignet ist. Diese Systeme benötigen logische Informationen in Form der sogenannten *features*, die aus den Daten der CAD-Modelle extrahiert werden müssen [Dixo89, Chan90]. So bilden in der Diskussion um die Rolle von solid modelling als Interface zwischen Konstruktion und Fertigung die *features* die Brücke zwischen der durch den Konstrukteur kreierten Geometrie und dem Arbeitsplan.

In der Literatur besteht bezüglich der präzisen Definition von *features* kein Konsens. Die überwiegende Mehrheit der Wissenschaftler dieser Domäne versteht unter *features* eine Abstraktion von lower-level Konstruktions- und Fertigungsinformationen [Dixo89]. So definieren John R. Dixon und John J. Cunningham in ihrem Papier [Cunn88] features als "*any geometric form or entity that is used in reasoning in one or more design or manufacturing activities,*". Tien-Chien Chang definiert in seinem Buch [Chan90] features als "*a subset of geometry on an engineering part which has a special design or manufacturing characteristic*.". J. J. Shah und M. T. Rogers definieren (informal) in ähnlicherweise in [Roge88] features als "*recuring patterns of information related to a part's description*.". Eine detailliertere Betrachtung bekannter Definitionen des Begriffs Features kann der interessierte Leser u. a. in [Klau91a oder Klau91b] finden. So sind die Features für einen Dreher zum Beispiel fertigungstechnische Bereiche auf dem Werkstück, mit dem er gewisse abstrakte Arbeitspläne (Skelettpläne) assoziiert; mittels den Features zerlegt sich also ein Dreher sein Werkstück in geeignete Bearbeitungsbereiche. In dem Sinne resultiert die Definition der Features in einer Aggregation von Flächen. Im Nachfolgenden ist ein kleiner Ausschnitt einer Featurebeschreibung eines Werkstückes zu sehen, die durch unseren Experten erstellt wurde.

In diesem Papier wird unter dem Begriff *Feature* ein auf den geometrischen und technologischen Daten eines Produktes basierendes Beschreibungselement verstanden, mit dem der jeweilige Experte von Konstruktion oder Fertigung in seinem Umfeld gewisse Informationen verbindet. (siehe auch [Klau91a] hierzu)



Diese Begriffsbestimmung ist an Tien-Chien Chang [Chan88] angelehnt und unterscheidet sich vor allem durch die explizite Betonung des Experten in seinem Umfeld. Das heißt insbesondere, daß alle Features durch den jeweiligen Experten beschrieben (definiert) werden, da in dieser Beschreibung sein Umfeld, wie etwa Maschinen, Werkzeuge oder auch Eigenarten von Maschinen und Werkzeugen, reflektiert wird und seine Ideen und Kreativität bezüglich seiner (Experten-)Arbeit, wie etwa besondere Tricks, enthalten sind; gerade letzteres stellt das sogenannte *know-how* einer Firma dar und darf in keinsterweise durch die Einführung von Expertensystemen verloren gehen. Daher obliegt es den featurebasierten Systemen im allgemeinen nur, eine Featurerepräsentationssprache zu definieren und nicht etwa die Features selbst.

Die Features werden einmal nach ihrer Art unterschieden als

- funktionale Features, wie etwa *Lagersitz* oder *O-Ring Nut*,
- qualitative Features, wie etwa *Stangenteil* oder *stabiles Teil*, oder

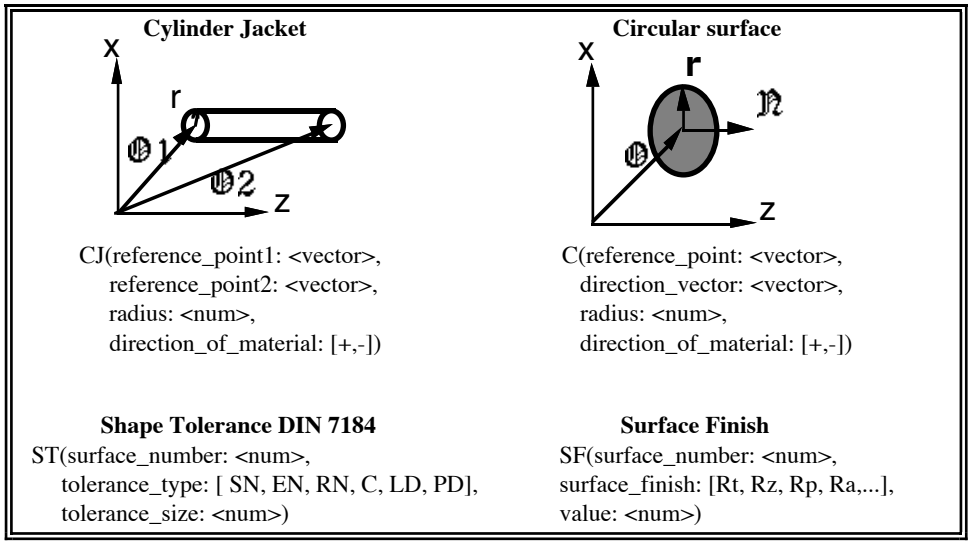
- gestalt (geometrische) Features, wie etwa *Schulter*, *Nut* oder *Bohrung*,
- atomare Features, wie etwa *Kreisring*, *Zylindermantelfläche* oder *Oberflächengüte*,

und zum anderen werden die Features nach der Anwendung unterschieden als

- Features der Konstruktion, wie etwa *Lagersitz* oder *Kuppler*
- Features der Fertigung:
  - Features des Drehen, wie etwa *Schulter* oder *Zapfen*,
  - Features des Fräsen, wie etwa *Absatz* oder *Tasche*,
  - Features des Bohren, wie etwa *Stufenbohrung* oder *Senkung*,
  - …
- …

Die Repräsentation von Features basiert bei den meisten featurebasierten Systemen auf der *Boundary Representation* (*B-rep*), das heißt, das die das Werkstück begrenzenden Oberflächen als kleinste geometrische Teile zur Definition von Features herangezogen werden.

In diesem Papier werden als *lower-level features* die geometrischen und technologischen Beschreibungselemente verstanden, wie sie in [Bern91] beschrieben sind. Nachfolgend sind einige kleine Beispiele dieser Repräsentation aufgeführt.

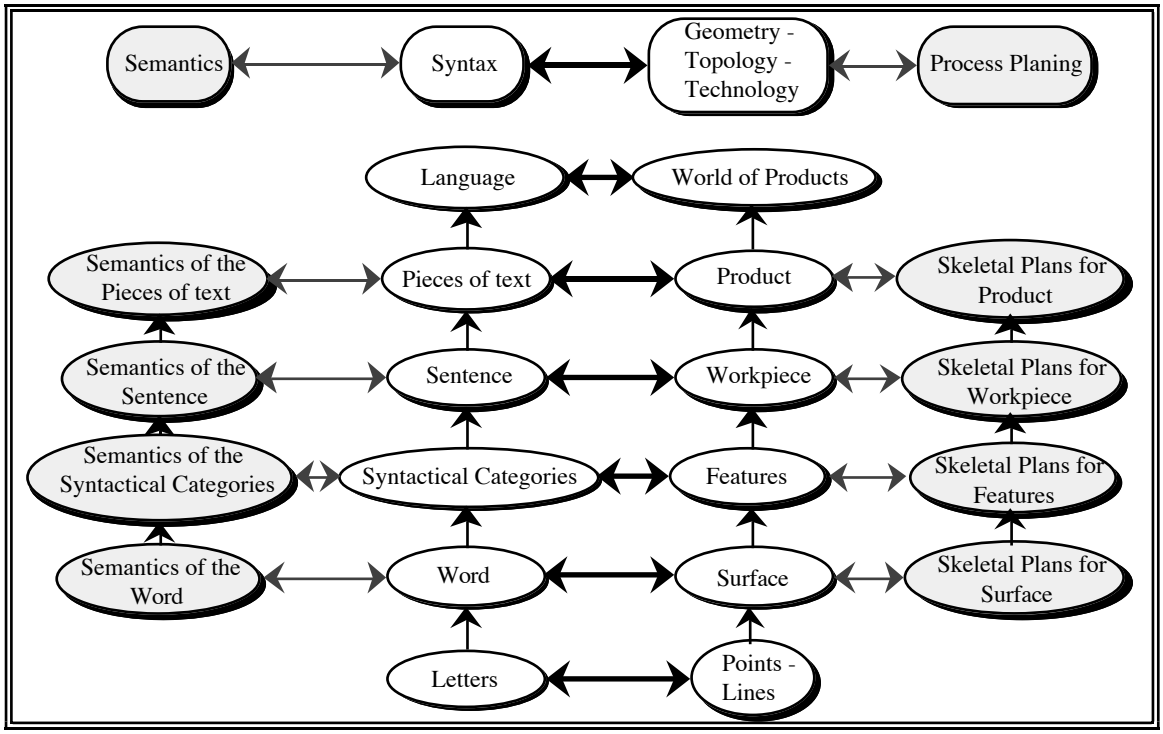


Die (*komplexen*) Features selbst, oder auch als *higher-level features* bezeichnet, definieren sich durch die atomaren Features bzw. durch (*komplexe*) Features.

Die Forschungen im Bereich der Featureerkennung sind vorwiegend im Bereich der featurebasierten Arbeitsplanung vorgenommen worden. Tony C. Woo in [Woo83] hingegen verwendet zum Beispiel die Featureerkennung zur Finite Elemente Analyse.

David C. Gossard und Hiroshi Sakurai stellen in [Goss90] einen Algorithmus zur Featureerkennung in 3D solid models vor, der basierend auf den *feature-Graphen*, die B-Rep basierte Subgraphen darstellen, Graphen matching benutzt. Zur Darstellung der Features wird aber im Gegensatz etwa zu [Fing90], keine Feature-Grammatik verwendet.

Innerhalb der Forschungen die um dieses Papier angesiedelt sind, sollen Features mittels parsing-Methoden erkannt oder auch generiert werden können. Dies wird vorallem dadurch ermöglicht, daß die Features in einer wohl definierten Grammatik, einer sogenannten attributieten Graph Grammatik, repräsentiert werden. Durch die Verwendung von Sorten und Heuristiken sollen praktikable Laufzeiten erreicht werden. Ermöglicht wird die Anwendung der parsing-Methoden durch eine Analogie der Feature-Definitionen mit formalen Sprachen. Diese Analogie ist im folgenden abgebildet. Das Problem der Featureerkennung läßt sich dann reduzieren auf das Problem, ein gegebenes Werkstück gemäß der gegebenen Feature-Grammatik zu parsen. Um nun die Nützlichkeit dieser Analogie mit Hilfe eines einfachen Prototypen eingeschrängt auf die Domäne der rotationssymmetrischen Werkstücke zu demonstrieren, wurde der Grammatikformalismus D-PATR so erweitert, daß mit ihm die Featurestruktur eines in TEC-REP (siehe hierzu [Bern91]) repräsentierten Werkstücks generiert werden kann.



5. Erweiterungen des D-PATR

Im Normalfall erwartet das D-PATR System als Eingabe die morphologischen Strukturen der zu analysierenden Wörter. Diese Strukturen sind in der grundlegenden Datenstruktur, den DAG's, repräsentiert. Um nun in dem D-PATR System Werkstücke handhaben zu können, werden durch ein dem D-PATR System vorgeschalteten Compiler aus der TEC-REP Darstellung der Flächen adequate DAG's erzeugt.

Insgesamt mußten in D-PATR nur die Funktionen bzw. Makros unify-atomic-macro, chart-parse und lookup-or-fake modifiziert bzw. erweitert werden, um FEAT-PATR zu erstellen. Dabei wurde durch Verwendung sogenannter *LISP*-Features (feat-rep) sowie des Prädikates feat-rep-p dafür Sorge getragen, daß D-PATR und FEAT-PATR ungestört nebeneinander existieren können.

Die eingelesenen Flächen werden intern zunächst geeignet aufbereitet, dann nach der z-Koordinate ihren linken Rändern sortiert, um sie in eine für den Parsevorgang geeignete Reihenfolge zu bringen. Diese Reihenfolge entspricht genau der Reihenfolge, in der die Flächen in dem Werkstück erscheinen. In der Funktion (transform-surface-rep) wird aus der TEC-REP Darstellung einer Fläche eine DAG-ähnliche Struktur erzeugt, die dem FEAT-PATR in der Funktion (look-up-or-fake) übergeben wird. Dabei werden die Technologieattribute jeweils den zugehörigen Flächen zugeordnet. Die adequaten DAG's selbst werden dann von FEAT-PATR aus den DAG-ähnlichen Strukturen erzeugt.

Die technologischen Grundelemente werden wie folgt behandelt: Erkannt werden die Elemente Thread (th), Surface Finish (sf) und Hardness (h), da diese Technologieattribute für die Featurerkennung mittels FEAT-PATR von besonderer Bedeutung sind. Diese drei Attribute können jeweils für eine Fläche nur einmal definiert sein und werden als neue Attribute den DAG's einer Fläche zugeordnet.

Ein DAG einer Fläche hat folgende Slots:

- **cat** - Der Typ der Fläche
- **geometry** - die geometrischen Daten
  - **radius** - der Radius
  - **direction_of_material** - die Materialrichtung
  - **reference_point1** - der erste Aufpunkt
  - **...**
- **id** - Die Identifikation einer Fläche

  - **name** - Name der Fläche
  - **num** - Nummer der Fläche
- **technology** - die technologischen Daten
  - **thread** - das Gewinde, falls definiert
  - **surface_finish** - Wert der Oberfläche, falls definiert
  - **hardness** - Wert der Materialhärte, falls definiert

Die Toplevel-Funktion chart-parse wird nicht wie in dem D-PATR System mit einer Liste von morphologischen Strukturen der Wörter, sondern einer Liste von geometrischen und technologischen Daten der Flächen aufgerufen. Mit einem kleinen Kunstgriff wird erreicht, daß FEAT-PATR dennoch nur die Namen der Flächen in seinen Ausgaben angibt: chart-parse ruft für jedes Wort check-phrase auf, dort wird wiederum lookup-or-fake aufgerufen. Diese Funktion gibt zwei Werte zurück, eine korrigierte Version des Eingabewortes, sowie den DAG des Wortes. Das modifizierte lookup-or-fake gibt zum einen den Namen der Fläche als Wort zurück, zum anderen generiert es aus den Flächendaten den DAG, der von check-phrase in den dag-cache eingetragen wird.

Das Makro unify-atomic-macro wurde geändert, um auch gleiche Zahlenwerte unifizieren zu können.

Gestartet wird das FEAT-PATR System, in dem mittels der Funktion select-and-read-wp eine Werkstückdatei geladen wird. Mit der Funktion save-one-parse kann danach ein Parseergebniss in einer Datei mit gleichem Namen wie die Werkstückdatei, ergänzt durch das Suffix features, abgespeichert werden. Dabei wird das Ergebniss mit der Funktion baum-aufbau von der internen FEAT-PATR Darstellung in eine vereinfachte Struktur transformiert.

6. Funktionale Erweiterungsmöglichkeit des Systems

Ursprünglich wurde das D-PATR-System zum Zwecke des Parsens natürlicher Sprache entworfen. Dies schließt jedoch seine Anwendbarkeit für andere Anwendungsdomänen nicht aus. Anstatt einen Satz einer natürlichen Sprache zu analysieren, kann das System ebenso einen geometrischen Körper parsen, sofern seine formale Beschreibung den Anforderungen der Syntax der Eingabe genügt. Um technischen Anwendungsbereichen gerecht zu werden, müssen die Spezifikationen der Regeln und Lexikoneinträge Vergleiche auf >,<,<=,>= zulassen. Diese Möglichkeit gilt es nun in dem System zu verwirklichen. Es folgt nun ein Integrationsvorschlag.

Jede Regel, die innerhalb der Spezifikationen Vergleichsoperatoren enthält, muß in einem extra dag-cache repräsentiert werden. Die Spezifikationen sollten in einem extra spec-array vermerkt werden. Aus der Eingabe für das System müssen die Teile der Spezifikationen entfernt werden, die mit einem Vergleichsoperator beginnen. Dies sollte schon beim Einlesen der Grammatik geschehen.

An zwei Stellen des D-PATR-Systems müßten Änderungen vorgenommen werden: In den Funktionen *try-to-extend* und *compatible-p* .

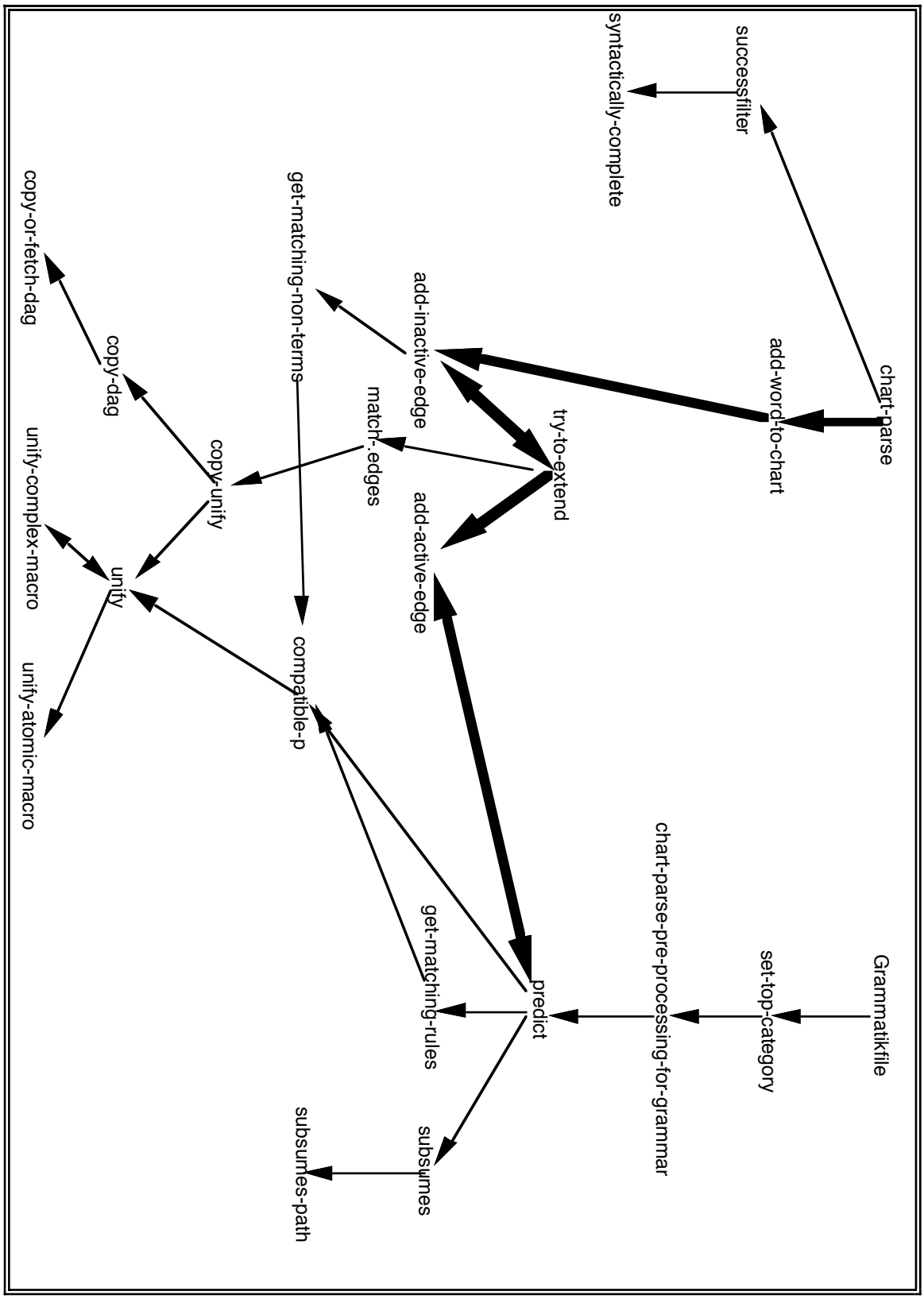
Während die Änderung bei *compatible-p* mehr aus Effizienzgründen sinnvoll wäre, ist die Änderung bei *try-to-extend* notwendig, da dort der bottom-up-parse initiiert wird. Die aus der ursprünglichen Grammatik entfernten sich auf Vergleichsoperatoren beziehenden Spezifikationen müssen jetzt wieder einbezogen werden. D.h. sie werden an dieser Stelle auf ihren logischenWahrheitswert geprüft.

Hinter der folgenden Programmzeile müssen Zusätze gemacht werden:

(setq new-feats (match-edges active inactive needed gap-flag)) .

new-feats enthält die Unifikation der Regel der aktiven Kante mit dem Kopf der Regel der inaktiven Kante. Dabei wurden die arcs der Regel der aktiven Kante, die keine Entsprechungen beim Kopf der Regel der inaktiven Kante haben, einfach übernommen.

Die ursprüngliche uninstantiierte Regel ist unter *rule* abgelegt. Nun muß nachgeschaut werden, ob *rule* im extra.dag-cache vermerkt ist. Ist dies nicht der Fall, so ist weiter nichts zuzusetzen. Andernfalls muß auf die entsprechenden Vergleiche im extra-spec-array zugegriffen werden.



Aufbaustruktur des D-PATR Systems

Ein Vergleich sehe syntaktisch so aus: (operator operand1 operand2). Der Zusatz müßte ungefähr so aussehen (Pseudocode):

```
(setq val1 (getvalue new-feats operand1))
(setq val2 (getvalue new-feats operand2))
(funcall operator val1 val2)  % wendet operator auf val1 und val2 an
```

Ist dieser Vergleich wahr, so ist die Gesamtanfangsbedingung der Funktion *try-to-extend* wahr, denn dieser Zusatz bildet das Ende der Anfangs-Bedingung. Es wird dann normal mit dem ursprünglichen Code der Funktion fortgefahren. Ist der Vergleich nicht erfüllt, so ist die Funktion beendet.

Analog ist bei der Funktion *compatible-p* vorzugehen. Das Ergebnis der Unifikation müßte in einer Variablen festgehalten werden, und die obigen Zusätze müßten auf diese Variable angewendet werden.

7. Warum ist FEAT-PATR zur Feature-Erkennung im allgemeinen ungeeignet ?

Die Eignung eines Grammatikformalismus läßt sich am besten dadurch bewerten, wie gut er den charakteristischen Eigenschaften der Features nachkommt, sprich wie stark er diese Charakteristika beim Parsen verwendet bzw. er überhaupt in der Lage ist, solche Charakteristika zu verarbeiten. Die detaillierte Beschreibung der Featurecharakteristika möge der interessierte Leser in [Klau91a] nachlesen. Nachfolgend wird FEAT-PATR im Hinblick auf diese Charakteristika bewertet.

- Features sind allgemeine Graphen:
  FEAT-PATR kann Graphen, die keine Sequenz bilden, nicht verarbeiten und läßt sich nicht dazu erweitern, da dies der Grundphilosophie des Systems widerspricht. Bei rotationssymmetrischen Werkstücken entspricht der Flächengraph genau einer Sequenz.

- Features sind durch Dimensionen definiert:
  FEAT-PATR erlaubt in der Definition der Feature-Regeln keine Prädikate $\neq$, $<$, $>$, $\leq$, etc. und keine Berechnung von Werten. Einziges Mittel zur Informationsgenerierung und zum Vergleich ist die Unifikation, sprich das =-Prädikat. Eine Erweiterung des Systems wäre möglich. Siehe hierzu Kapitel 6.

- Features können interagieren:
  FEAT-PATR erlaubt keine direkte Repräsentation der Featureinteraktion. Feature-Regeln definiert durch interagierende Features, wie etwa die Nut, die durch l-Schulter und r-Schulter definiert sein kann, müssen transformiert werden, um die "Interaktion" aufzulösen, sprich allle Kombinationen der interagierenden Features (l-Schulter, r-Schulter) bilden eine neue Regel (der Nut). Innerhalb der Relationen über die Attribute kann die Interaktion wieder kodiert werden. Dieses Verfahren ist sehr ineffizient: es werden von einem Graph mindestens drei Graphen auf einer Ebene erzeugt: einmal wo die l-Schulter erkannt wird, dann wo die r-Schulter erkannt wird und dann wo die Nut erkannt wird. FEAT-PATR läßt sich zur direkten Verarbeitung von Interaktionen nicht erweitern, da es seiner Grundphilosophie resultierend aus der Unifikation widerspricht. Der Parsing-Algorithmus müsste ganz neu geschrieben werden.

- Features sind meist durch sehr viele ähnliche Regeln definiert:

FEAT-PATR arbeitet mit Breitensuche und generiert alle möglichen Parse. Gefordert ist eine Tiefensuche (aus Effizienzgründen) Von 100 (00) möglichen Parses sind ca. 80% relativ gleichwertig resultierent aus den vielen ähnlichen Regeln. Eine Erweiterung von FEAT-PATR ist nicht möglich, da es wie oben erwähnt, auch hier der Grundphilosophie widersprechen würde.

- Definitionen der Features beinhalten auch Heuristiken betreffend ihrer Anwendung:
  FEAT-PATR kann keine Heuristiken verarbeiten, die sich in einer Partialordnung über den Regeln äussern könnte. Eine Erweiterung von FEAT-PATR ist nicht möglich, da es wie oben erwähnt, auch hier der Grundphilosophie widersprechen würde.

- Features können durch fragmentierte Bereiche definiert sein:
  FEAT-PATR kann keine fragmentierten Regeln verarbeiten. Hierzu muß in den Regeln expliziet alle möglichen Zwischenteile zwischen den Fragmenten aufgeführt werden. Eine Erweiterung des FEAT-PATR Systems ist hier ebenfalls nicht möglich.

- Features können in Abhängigkeit ihres Kontextes definiert sein:
  FEAT-PATR ist nur für kontextfreie Grammatiken geeignet. Durch aufstellen von Hypothesen lassen sich jedoch die kontextsensitiven Feature-Regeln von FEAT-PATR verarbeiten. Eine Erweiterung von FEAT-PATR ist nicht möglich.

8. Literatur

[Bern91] Bernardi, A., Klauck, C., and Legleitner, R.: *TEC-REP: Repräsentation von Geometrie- und Technologieinformationen*. Dokument, D-91-07 Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Postfach 20 80, D-6750 Kaiserslautern, june, 1991.

[Chan88] Chang, G.J. and Henderson, M.R.: FRAPP: Automated Feature Recognition and Process Planning from Solid Model Data. In *International Computers in Engineering Conference and Exhibition,* july/august 1988.

[Chua91] Chuang, S.H. and Henderson, M.R.: *Compound Feature Recognition by Web Grammar Parsing*. Research in Engineering Design 2 (1991), 147-158.

[Chan90] Chang, T.C.: *Expert Process Planning for Manufacturing,* Addison-Wesley (1990).

[Cunn88] Cunningham, J.J. and Dixon, J.R.: Designing with features: the origin of features. In *International Computers in Engineering Conference and Exhibition,* july/august 1988, pp. 237-243.

[Dixo89] Dixon, J.R. and Finger, S.: *A Review of Research in Mechanical Engineering Design. Part II :Representations, Analysis, and Design for the Life Cycle*. Engineering DesignSpringer-Verlag New York Inc. (1) 2 (1989), 121-137.

[Fing90] Finger, S.: Parsing Features in Solid Geometric Models. In *ECAI90,* 1990.

[Goss90] Gossard, D.C. and Sakurai, H.: *Recognizing Shape Features in Solid Models*. IEEE Computer Graphics and Applications (1990), 22-32.

[Kartt86] Karttunen L.: *D-PATR: A Development Environment for Unification-based Grammars*. Stanford 1986

[Klau91a] Klauck, C., Bernardi, A., and Legleitner, R.: FEAT-REP: Representing Features in CAD/CAM. Research Report RR-91-20, DFKI GmbH, 1991.

[Klau91b] Klauck, C., Bernardi, A., and Legleitner, R.: FEAT-REP: Representing Features in CAD/CAM. In *IV International Symposium on Artificial Intelligence:Applications in Informatics,* 1991.

[Kypr80] Kyprianou, L.K.: *Shape Classification in Computer-Aided Design*, Ph.D. dissertation, Christ`s College, University of Cambridge, P.O Box 1745 Nicosia Cyprus, July 1980.

[Roge88] Rogers, M.T. and Shah, J.J.: *Expert form feature modelling shell*. Computer Aided Design (20) 9 (november 1988), 515-524.

[Woo83] Woo, T.C.: *Interfacing Solid Modeling to CAD and CAM: Data Structures and Algorithms for Decomposing a Solid*. ASME Computer-Integrated Manufacturing (1983), 39-45.

9. Anhang

Nachfolgend ist die von FEAT-PATR verwendete Feature-Grammatik abgebildet, die aus Feature-Definitionen eines Experten gewonnen wurde. Der Experte ist ausgebildeter Dreher (3 Jahre Lehrzeit) und hat 1 Jahr praktische Erfahrung in diesem Beruf.

Deutsches
Forschungszentr
um
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608
Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter pub/Publications bezogen werden.
Die Berichte werden, wenn nicht anders gekenn-zeichnet, kostenlos abgegeben.

DFKI Research Reports

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

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

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

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

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

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

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

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or via anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.
The reports are distributed free of charge except if otherwise indicated.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

RR-94-03
Gert Smolka: A Calculus for Higher-Order Concurrent Constraint Programming with Deep Guards
34 pages

RR-94-05
Franz Schmalhofer, J.Stuart Aitken, Lyle E. Bourne jr.: Beyond the Knowledge Level: Descriptions of Rational Behavior for Sharing and Reuse
81 pages

RR-94-06
Dietmar Dengler: An Adaptive Deductive Planning System
17 pages

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

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

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

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

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

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

RR-94-14
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Towards a Sharable Knowledge Base on Recyclable Plastics
14 pages

RR-94-15
Winfried H. Graf, Stefan Neurohr: Using Graphical Style and Visibility Constraints for a Meaningful Layout in Visual Programming Interfaces
20 pages

RR-94-16
Gert Smolka: A Foundation for Higher-order Concurrent Constraint Programming
26 pages

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

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

RR-94-20
Christian Schulte, Gert Smolka, Jörg Würtz:
Encapsulated Search and Constraint Programming in Oz
21 pages

RR-94-31
Otto Kühn, Volker Becker, Georg Lohse, Philipp Neumann:
Integrated Knowledge Utilization and Evolution for the Conservation of Corporate Know-How
17 pages

RR-94-33
Franz Baader, Armin Laux:
Terminological Logics with Modal Operators
29 pages

DFKI Technical Memos

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

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

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

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

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

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

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

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

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

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

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

DFKI Documents

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

D-94-10
F. Baader, M. Lenzerini, W. Nutt, P. F. Patel-Schneider (Eds.): Working Notes of the 1994 International Workshop on Description Logics
118 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-94-11
F. Baader, M. Buchheit, M. A. Jeusfeld, W. Nutt (Eds.):
Working Notes of the KI'94 Workshop: KRDB'94 - Reasoning about Structured Objects: Knowledge Representation Meets Databases
65 Seiten

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