Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Document
D-93-26

Unterstützung des Experten bei der Formalisierung von Textwissen

INFOCOM
Eine interaktive Formalisierungskomponente

Frank Peters

Dezember 1993

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, 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, SEMA Group, and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

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.

Friedrich J. Wendl
Director

Unterstützung des Experten bei der Formalisierung von Textwissen

INFOCOM - Eine interaktive Formalisierungskomponente

Frank Peters

DFKI-D-93-26

Diese Arbeit wurde finanziell unterstützt durch das Bundesministerium für Forschung und Technologie (FKZ ITW-9304/3).

Unterstützung des Experten bei der Formalisierung von Textwissen

INFOCOM

Eine interaktive Formalisierungskomponente

Frank Peters
Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)
67608 Kaiserslautern
Germany
e-mail:
peters@dfki.uni-kl.de

Zusammenfassung

Ein in der Wissensakquisition grundlegendes Problem ist das Formalisieren informalen Wissens. Für die Lösung dieses Problems bieten sich verschiedene Wege an, die sich nicht zuletzt durch den Grad ihrer Automatisierung unterscheiden. In dieser Arbeit soll das System INFOCOM vorgestellt werden, welches einen Experten der zu bearbeitenden Domäne beim formalisieren von Textwissen unterstützt. Der Vorgang des Formalisierens ist in verschiedene Phasen unterteilt, die zum einen automatisch durch einen regelgesteuerten Füllalgorithmus, zum anderen manuell im interaktiven Dialog mit dem Experten durchgeführt werden. Ein Wissensingenieur hat die Aufgabe, den Füllalgorithmus durch Eingabe von Regelsystemen zu konfigurieren und den Experten bei seiner Arbeit zu unterstützen.

Abstract

A fundamental problem with knowledge acquisition is the formalisation of informal knowledge. In order to solve this problem there are various ways which, last but not least, differ by the degree of their automatic control. In this document the system INFOCOM is to be presented which supports domain-experts while formalising text-knowledge. The process of formalisation is divided into different phases which are executed automatically by a rule-controlled filling-algorithm on the one hand and manually through an interactive dialog with the expert on the other hand. The task of the knowledge engineer is the configuration of the filling-algorithm by creating rule-systems. In addition to this, he has to support the work of the expert.

Inhaltsverzeichnis

1. Einleitung

Ein in der Wissensakquisition grundlegendes Problem ist das Formalisieren informalen Wissens. Je nachdem, welche Vorraussetzungen und Hilfsmittel für die Lösung dieses Problems gegeben sind, bieten sich verschiedene Vorgehensweisen an. In Kapitel 3 werden Lösungsansätze diskutiert, welche sich nicht zuletzt durch den Grad ihrer Automatisierung beim Formalisieren von Wissen unterscheiden.

In dieser Arbeit soll die Vorgehensweise der interaktiven Formalisierungskomponente **INFOCOM (Interactive Formalisation Component)** beim Formalisieren informalen Wissens vorgestellt werden. In INFOCOM werden *Wissenseinheiten* verarbeitet, welche elementare Teile des Wissens einer *Domäne* (z.B. Maschinenbau) in informaler, semiformaler bzw. formaler Art repräsentiert darstellen.

Für eine *informale Wissenseinheit*, welche aus einem bis wenigen natürlichsprachlichen (dekontextualisierten) Sätzen besteht, wird in INFOCOM eine formale Wissenseinheit produziert, welche einer vorgegebenen Zielrepräsentationssprache entspricht. Dabei wird für eine informale Wissenseinheit als erstes eine Menge von *Schablonen* (*abstrakte Templates*) vorgeschlagen, welche die für die Wissenseinheit entsprechende Formalisierungsvorgabe in der Zielrepräsentationssprache darstellen.

Der Vorgang des Formalisierens ist in INFOCOM in drei Phasen unterteilt::

i.) Auswahl eines abstrakten Templates

ii.) (Teilweises) Füllen eines abstrakten Templates

iii.) Editieren eines gefüllten Templates (falls notwendig)

Die Benutzer des Systems teilen sich in zwei Gruppen auf:

Der *Anwender* ist vorzugsweise ein Fachmann (*Experte*) der zu bearbeitenden Domäne, welcher die in den Phasen 1 und 3 manuell zu bewältigenden Arbeiten durchführt. Aufgrund seiner Expertise kann er die Informationen und Zusammenhänge der informalen Wissenseinheit erkennen und somit ein der Wissenseinheit entsprechendes, abstraktes Template auswählen (Phase1), bzw. mögliche ungefüllte oder unkorrekt gefüllte Passagen eines (teilweise) gefüllten Templates erkennen und editieren (Phase 3).

Da ein Experte im allgemeinen kein Computerspezialist ist, wird dieser durch einen *Wissensingenieur* (*KE* für *Knowledge Engineer*) unterstützt. Neben einer anfänglichen Einführung des Experten in die Bedienung des Systems soll der KE die *Konfiguration* des in Phase 2 in Erscheinung tretenden *Füllalgorithmus* vornehmen.

Daraus ergeben sich für die Entwicklung von INFOCOM folgende Anforderungen:

- **Unterstützung des Experten beim Füllen**
  Der Anwender des Systems ist ein Domänenexperte und kein Computerspezialist. Da ein Experte (im allgemeinen) nicht über die für eine Formalisierung erforderlichen Kenntnisse verfügt, sollte er in höchstem Maße bei seiner Arbeit unterstützt werden. Dies soll bei den nachfolgenden Punkten stets beachtet werden.

- **Konfigurierbarkeit des Füllalgorithmus**
  Ein abstraktes Template läßt sich in Passagen, oder *Muster* (*Pattern*) unterteilen, welche mit den Wörtern der Wissenseinheit gefüllt werden sollen. Das Wissen, welches ein Experte beim manuellen Füllen dieser Füllpattern anwenden würde, muß dem Füllalgorithmus zugänglich gemacht werden. Hierfür wird der Füllalgorithmus durch *Füllregeln* gesteuert, die in *Regelsystemen* zusammengefaßt werden (Kapitel 5.2.). Füllregeln beschreiben *Eigenschaften* für diese Pattern und testen während des Füllvorgangs die zur Verfügung stehenden Elemente auf das Vorhandensein solcher Eigenschaften. Da es oft schwierig, bzw. auch unmöglich ist, genau die gewünschten Eigenschaften anzugeben, welche die für ein Pattern vorgeschlagenen Elemente besitzen sollten, wird dieses Problem von zwei Seiten angegangen:
  i.) Für ein Element, das für ein Pattern vorgeschlagen werden soll, werden zum einen die Eigenschaften angegeben, die es *auf jeden Fall* haben muß, und
  ii.) zum anderen die Eigenschaften, die es auf *keinen Fall* haben darf.
  Technisch wird dies durch *konstruktive* und *destruktive* Füllregeln realisiert (Kapitel 5.2.1.). Durch konstruktive Füllregeln werden die unter i.) genannten Elemente bestimmt und mit einer *Bewertung* belegt (Bewertungsregeln). Destruktive Füllregeln entfernen dagegen die Elemente, welche die unter ii.) genannten Eigenschaften aufweisen (Filtrationsregeln).
  Durch das Definieren neuer Füllregeln, bzw. neuer Regelsysteme, läßt sich der Füllalgorithmus *während* der Erstellung der Zielrepräsentation konfigurieren und somit optimieren.

- **Erweiterbarkeit des Systems**
  Die Eigenschaften, die für ein zu füllendes Pattern durch eine Füllregel charakterisiert werden, werden durch logische Verknüpfung von *Prädikaten* beschrieben (Kapitel 5.1.2.). Prädikate werden direkt in einer Programmiersprache (LISP) formuliert, so daß sie auf die Funktionen, die auf dieser Ebene definiert sind, zugreifen können. Um eine Erweiterbarkeit des Systems zu gewährleisten, kann die Menge dieser Funktionen um die Funktionalität externer Moduln erweitert werden (z.B. die Funktionalität einer Morphologie). Hierfür werden beim Laden des

Systems diese Moduln und die für sie vordefinierten *Interfaces* mitgeladen, welche eine Anpassung an das System erlauben (Anhang C).

- **Kapselung von Informationen**
  Um ein geeignetes Arbeiten mit INFOCOM zu gewährleisten, soll jedem Benutzer nur die für ihn relevanten Informationen dargelegt werden. Im Gegensatz zu [PuertaEgar+92], bei denen dem KE und dem Experte jeweils ein eigenes Tool bereitgestellt wird, soll die Benutzerführung in zwei getrennte Modi aufgeteilt: Den *Expert-* und den *KE-Mode*. Im Gegensatz zum KE-Mode, in dem die gesamte Funktionalität von INFOCOM bereitgestellt wird, beschränkt sich das Arbeiten im Expert-Mode "lediglich" auf das Füllen abstrakter Templates mittels eines bereits vordefinierten Regelsystems.

- **Vermittlung von Informationen**
  Ein bekanntes Problem bei der Entwicklung von KA-Methoden (KA für Knowledge Acquisition) ist der sogenannte "Flaschenhals" (Bottleneck) der Wissensakquisition. Dieses Problem kennzeichnet nicht nur für die Frage, wie man für eine Wissensbasis das relevante Wissen einer Domäne erhält, sondern auch, wie man relevantes Domänenwissen automatisch formalisieren kann. Für die Konfiguration eines effizienten Füllalgorithmus ist es daher unerläßlich, daß die Personen der beiden Benutzergruppen Kenntnisse über die für sie unbekannte Domäne erlangen können.
  Das betrifft zum einen den KE, der aufgrund von Vorschlägen seitens des Experten Regeln zum Füllen der abstrakten Templates definieren muß. Zum anderen betrifft es den Experten. Wird diesem die Möglichkeit geboten, sich in den zum Erstellen eines Regelsystems notwendigen Formalismus einzuarbeiten, so kann der Experte aufgrund seiner Expertise eigene Regelsysteme zur Steuerung des Füllalgorithmus eingeben. In Kapitel 5.2.2. werden hierfür verschiedene Sichtweisen, sogenannte *Views einer Füllregel* eingeführt, welche eine Füllregel auf verschiedenen Abstraktionsebenen darstellen.

Im zweiten Teil dieser Arbeit wird ein den oben genannten Punkten entsprechender Füllalgorithmus vorgestellt, der in einer geeigneten Benutzeroberfläche eingebettet ist. Neben Dialogfenstern, die das Füllen der abstrakten Templates unterstützen, werden auf dieser Oberfläche verschiedene *Werkzeuge* (*tools*) bereitgestellt, welche das Editieren von Prädikaten, Regeln und Regelsystemen erlauben. Kapitel 6 beschreibt die Bedienung von INFOCOM beim Füllen abstrakter Templates und das Erstellen eines Regelsystems mit Hilfe der dafür vorgesehenen Tools (Kapitel 6.2.2.).

2. Voraussetzungen

INFOCOM ist als Formalisierungskomponente in der Wissensakquisitionsmethode COKAM+ (Case Oriented Knowledge Acquisition Method from Text) integriert. Daneben nutzt INFOCOM die Funktionalität der Morphologie MORPHIX. Die folgenden Unterkapitel sollen COKAM+ und MORPHIX kurz beschreiben und einen Überblick der in dieser Umgebung vorhandenen Hilfsmittel geben.

2.1 Die Wissensakquisitionsmethode COKAM+

COKAM+ [Schmidt92] ist Teil der integrierten KA-Methode des ARC-TEC-Projektes, welche ausführlich in [SchmalhoferKühn+91] und [SchmalhoferSchmidt91] beschrieben wird. Der Vorgang des Erhebens von Wissen wird in COKAM+ in verschiedenen, aufeinander aufbauenden Phasen vorgenommen [SchmidtPeters+93]:

2.1.1. Wissenseinheiten und das Modell der Expertise

Während der Erstellung der Zielrepräsentation werden in COKAM+ Wissenseinheiten ihrem Kontext entsprechend einer adequaten Kategorie des Modells der Expertise zugeordnet. Eine solche Kategorie wird als *Expertisenview* oder kurz *View* bezeichnet. Ein View beschreibt die Klassen, welche im Sinne von KADS [WielingaSchreiber+92] eine Ein- oder Ausgabeklasse in der domänenspezifischen Inferenzstruktur darstellen.

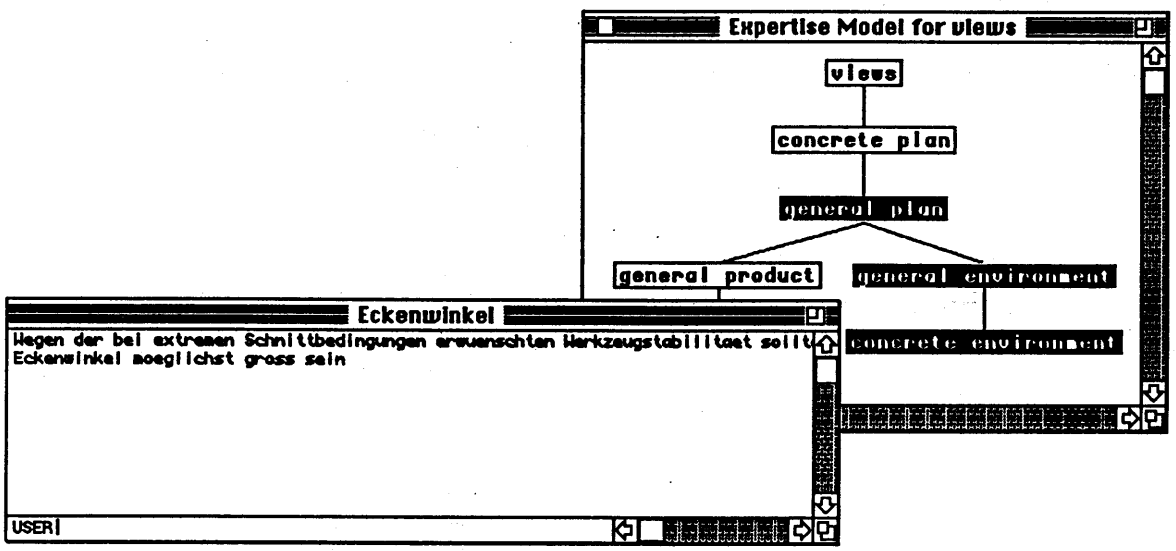


Abbildung 2.1: Eine Wissenseinheit mit ihren assoziierten Views

Die Abbildung 2.1. zeigt eine Wissenseinheit,die den Titel "Eckenwinkel" hat, und das Modell der Expertise, in welchem die mit der Wissenseinheit assoziierten Views markiert

2.1.2. Zuordnung von Wissenseinheiten zu Fällen

COKAM+ ist eine *fallorientierte* Wissensakqusitionstmethode, in der mit Hilfe von *Fällen* Wissen aus Texten extrahiert wird. Ein Fall besteht aus einem in der Domäne vorkommenden *Problem* und einer *Lösung*, die für dieses Problem bereits gefunden wurde. In COKAM+ wird ein solcher Fall durch eine eingescannte CAD-Graphik dargestellt. Abbildung 2.2 (links) zeigt die Problembeschreibung eines zu fertigenden Werkstücks, die Operatoren, durch die das Werkstück gefertigt wird und deren Reihenfolge.

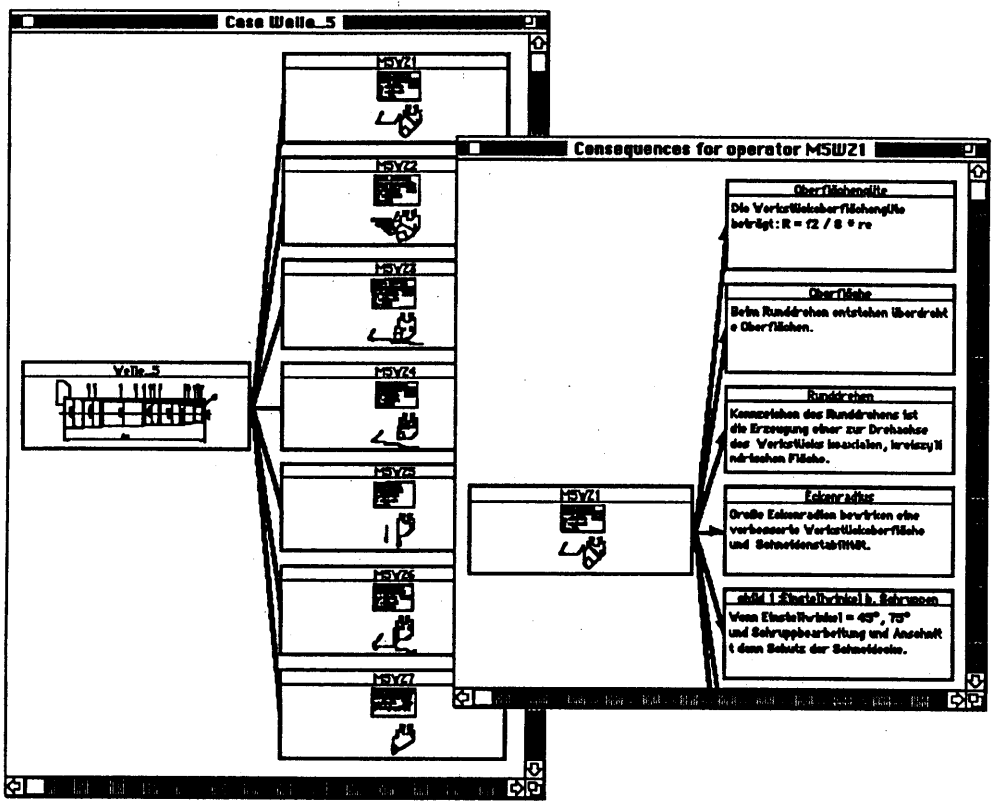


Abbildung 2.2: Ein Fall mit seinen Operatoren.
Wissenseinheiten beschreiben hier die Nachbedingungen des Operators M5WZ1.

Wissenseinheiten dienen hierbei als Vor- und Nachbedingungen solcher Operatoren. (Abbildung 2.2, rechts: Nachbedingungen des Operators M5WZ1).

2.1.3. Schablonen (abstrakte Templates)

Für die formale Darstellung des erhobenen Textwissens verwendet COKAM+ eine eigene Zielrepräsentationssprache. Aufgrund einer solchen, oben genannten Kategorie von Expertisenviews, können für eine informale Wissenseinheit Schablonen (abstrakte Templates) vorgeschlagen werden, welche ein Algorithmus [Huf94, in press] automatisch generiert. Es stehen vier konkrete, von der Wahl einer Zielrepräsentationssprache abhängige Typen abstrakter Templates zur Verfügung:

- **precondition- und consequence-templates** (condition templates)
  Precondition- und Consequence-Templates besitzen eine Ableitungsrichtung. Sie beschreiben Vor- bzw. Nachbedingungen eines Operators in der Zielrepräsentationssprache der KA-Methode.

- **abstraction-refinement-rules** (constraint template)
  Abstraction-Refinement-Rules dienen zur Verfeinerung des erhobenen Wissens, bzw. der Wissenseinheiten. Sie besitzen im Gegensatz zu den oben genannten Templatetypen keine Ableitungsrichtung.

- **gewichtete Constraints** (constraint template)
  Den vierten Typ abstrakter Templates bilden die sogenannten *gewichteten Constraints*, welche Formalisierungsvorgaben in der konkreten Zielrepräsentationssprache CONTAX [Tolzmann92] sind und in einer Parallelversion von COKAM+ untersucht werden. Sie sollen in dieser Arbeit nicht näher behandelt werden.

Die folgende Abbildung zeigt eine Wissenseinheit, für die ein abstraktes Precondition-Template durch einen Algorithmus generiert wurde. Die hier dargestellten Symbole CPLAN, ENV und SPLAN sind beispielsweise Platzhalter und beschreiben zu füllenden Passagen dieses abstrakten Templates. Sie stehen für die Views "concrete plan", "environment" und "general plan" des Modells der Expertise. Darunter ist ein gefülltes Template dargestellt, das durch einen Experten manuell gefüllt worden ist.

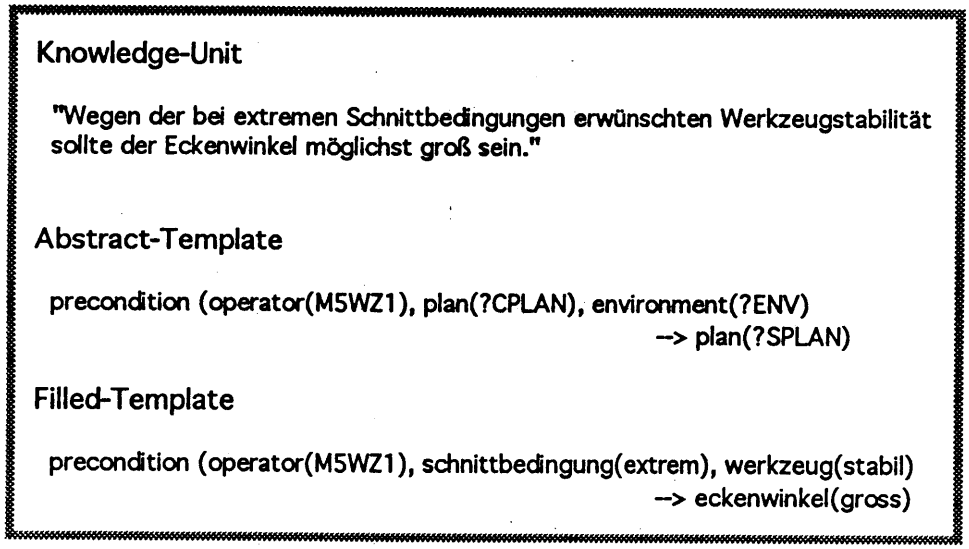


Abbildung 2.3.: Eine Wissenseinheit mit abstraktem und gefülltem Template.

2.2. Die Morphologie MORPHIX

MORPHIX [FinklerNeumann86] ist ein hochportabler Lemmatisierungsmodul für das Deutsche, der von Wolfgang Finkler und Günter Neumann am Fachbereich Künstliche Intelligenz - Wissensbasierte Systeme der Universität des Saarlandes implementiert wurde.

Lemmatisierungskomponenten dienen der Reduzierung von Lexika in natürlichsprachlichen Systemen, indem flektierte Wortformen auf kanonische Formen zurückgeführt werden und ihre grammatikalische Information bestimmt wird. In MORPHIX werden durch einen sogenannten GRIN-BAUM (**G**rammatikalischer **In**formations Baum), in dem der Zusammenhang zwischen Flexionsendung und zugehöriger grammatikalischer Information kodiert ist, Wortformen regelgeleitet bearbeitet. Kann eine Wortform nicht bearbeitet werden, so kann durch einen interaktiven Klärungsdialog eine Lexikonerweiterung vorgenommen werden.

In INFOCOM wird MORPHIX als unabhängiger Modul für das automatische Füllen abstrakter Templates eingesetzt. Bei diesem Vorgang wird das Auftreten des oben genannten Klärungsdialogs unterdrückt, da es sich hier als störend erweist. Es wird also davon ausgegangen, daß vor Beginn des Füllvorgangs das Lexikon für die Analyse der Wörter einer Wissenseinheit vorbereitet ist.

MORPHIX ist bereits in der KA-Methode COKAM+ eingebunden und wird zur Strukturierung nach dem Modell der Expertise eingesetzt [Huf94, in press]. In dieser Arbeit soll anhand des Lemmatisierungsmoduls gezeigt werden, wie in INFOCOM durch Erstellung von Interfaces die Funktionalitäten zusätzlicher Moduln für den automatischen Vorgang des Füllens abstrakter Templates genutzt werden können.

3. Verschiedene Vorgehensweisen beim Formalisieren von Textwissen

Bei der Formalisierung von Textwissen sollen für die natürlichsprachlichen Ausdrücke eines Textes, welche relevantes Wissen der Domäne beinhalten, Ausdrücke in einer formalen Sprache bestimmt werden. Vergleicht man hinsichtlich dessen eine natürliche Sprache mit einer formalen, so ergeben sich aus der präzisen Syntax und der Semantik einer formalen Sprache verschiedene Probleme [SchmidtWetter89].

Ein Autor schreibt einen (wissenschaftlichen) Text aufgrund des Gefühls, welches er für die

Menge der *korrekten* Sätze einer natürlichen Sprache besitzt. Diese Menge weicht jedoch von der Menge der Sätze ab, die während des automatischen Formalisierens durch einen Parser von irgend einer existierenden Grammatik akzeptiert werden. Daneben mißt der Autor jedem Wort, das er benutzt, eine ganz bestimmte Bedeutung zu, so daß es keine 1 zu 1 Beziehung zwischen den natürlichsprachlichen Ausdrücken und deren formalen Abbildungen gibt, welche die intendierte Bedeutung reflektieren.

Für die Lösung dieses Problems existieren verschiedene Ansätze, die sich sowohl in der Vorgehensweise, als auch durch den Grad ihrer Automatisierung unterscheiden.

Textwissen kann mit Hilfe *natürlichsprachlicher Dialoge* formalisiert werden. In dem System KALEX[1] [SchmidtWetter89], werden Wissenseinheiten durch einen Experten in natürlichsprachlichen Sätzen (Deutsch) eingegeben und anschließend automatisch formalisiert. Hierbei versucht das System einen Satz in eine interne Darstellung, die sogenannte DRS[2] [Kamp81], zu übersetzen. Wird eine solche DRS gefunden, so kann der Experte diese verifizieren und gegebenenfalls dem Kontext des Satzes entsprechend editieren. Anderenfalls wird der Experte aufgefordert, den eingegebenen Satz zu modifizieren, so daß eine Analyse durch das System erfolgen kann. In KALEX werden demnach durch einen Experten nacheinander Wissenseinheiten in einem interaktiven Dialog eingegeben und formalisiert, wobei der Experte hier die formale Sprache der DRS vielleicht nicht beherrschen, jedoch zumindest *benutzen* muß.

[1] Knowledge Acquisition for LEX

In [LiddyJörgenson+92] wird eine NLP-Komponente (Natural Language Processing Component) für Expertensysteme beschrieben, welche für einen Text eine umfassende, semantische Repräsentation erstellt. Hierzu wird durch einen sublanguage-Prozessor ein strukturiertes Wissensrepräsentationsformat kreiert, in dem jede Einheit der Daten des Textes auf die semantische Rolle hin untersucht wird, die sie in dem Text spielt. Anschließend wird der so semantisch durcharbeitete Text in strukturierte Informationen unterteilt. Hierbei wird das Untersuchen der Semantik individueller Wörter und/oder Wortkombinationen dem Untersuchen der grammatikalischen Kathegorie eines einzelnen Wortes vorgezogen.

In INFOCOM soll das Formalisieren einer informalen Wissenseinheit in einem interaktiven Dialog erfolgen. Hierbei soll das System dem Experten für ein gegebenes, abstraktes Template (in möglichst kurzer Zeit !) eine Reihe von Vorschlägen liefern, die von einem Füllalgorithmus erzeugt und bewertet werden. Damit ist von vornherein der Fall eingeplant, daß der *beste* Vorschlag des Systems nicht der gewünschte sein muß. Tritt dieser Fall ein, so soll, im Gegensatz zu der Vorgehensweise in KALEX, die Korrektur des Ergebnisses nicht durch das Verändern der Wissenseinheit erreicht werden. Vielmehr soll dann durch das Modifizieren diverser Fülloptionen versucht werden, eine bessere Annäherung an das gewünschte Ergebnis zu erhalten. Durch ein restriktives Editieren des Ergebnisses sollen auch hier mögliche semantische Lücken zwischen der informalen Wissenseinheit und ihrer formalen Abbildung geschlossen werden, ohne die syntaktische Korrektheit des Ergebnisses hinsichtlich der Zielrepräsentationssprache zu beeinträchtigen.

Die Vorschläge sollen durch einen Füllalgorithmus regelgesteuert erzeugt werden. Im Gegensatz zu [LiddyJörgenson+92], wo für einen gesamten Text eine umfassende, semantische Repräsentation erstellt wird, sollen durch sogenannte *Füllregeln* einzelne Wörter oder Wortkombinationen auf verschiedene *Eigenschaften* hin untersucht werden. Dazu gehört, neben der Untersuchung der grammatikalischen Kathegorie eines Wortes, beispielsweise auch der Test, ob ein Wort mit einer bestimmten Klasse der Domäne assoziiert ist (Schlüsselwort).

4. Einteilung der Formalisierung in drei Phasen

Im folgenden soll die Vorgehensweise des hier vorgestellten Systems bei der Formalisierung einer informalen Wissenseinheit beschrieben werden.

Der Experte soll bei der Formalisierung von informalen Wissenseinheiten in hohem Maße unterstützt werden. Hierzu soll ihm ein Dialog geboten werden, der diesen Vorgang in einzelne Phasen unterteilt darstellt:

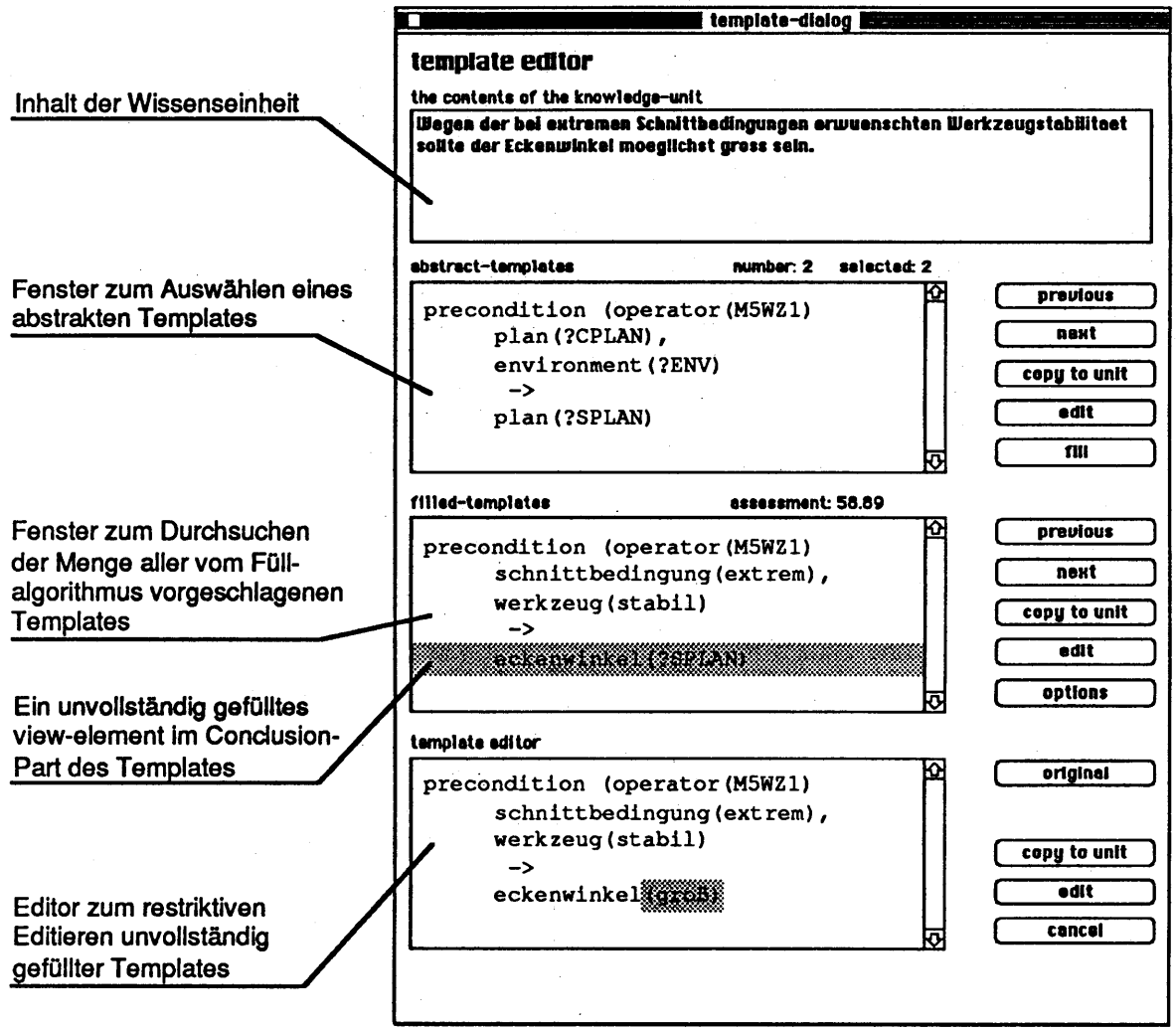


Abbildung 4.1: Hauptdialog zum Füllen abstrakter Templates.

Um den Kontext der ausgewählten Wissenseinheit vor Augen zu haben, wird deren Inhalt in dem obersten Fenster des Dialogs dargestellt. Unter diesem Fenster sind zwei Auswahlfenster angeordnet, die das Parsen und Auswählen der abstrakten, bzw. vom Füllalgorithmus vorgeschlagenen Templates erlaubt. Aus diesen Auswahlfenstern können die Templates

entweder in den Unit-Stack[3], oder in einen Editor kopiert werden, in welchem die nicht-, bzw. unkorrekt gefüllten Passagen des Templates editiert werden können. Die einzelnen Phasen werden in den nachfolgenden drei Punkten beschrieben.

- **Auswahl eines abstrakten Templates**
  Für eine ausgewählte Wissenseinheit werden durch einen Algorithmus verschiedene Templates generiert und aufgelistet. Der Experte wählt mit Hilfe seiner Expertise ein geeignetes Template aus, welches als Formalisierungsvorgabe der zu bearbeitenden Wissenseinheit dient, und beauftragt den Füllalgorithmus, dieses zu füllen.

- **Füllen eines abstrakten Templates**
  Für ein ausgewähltes Template liefert der Füllalgorithmus eine Menge gefüllter Templates. Im besten Fall ist in dieser Menge ein *vollständig* und *korrekt* gefülltes Template enthalten. Der Experte wählt dieses aus und produziert so eine neue, *formale* Wissenseinheit. Wurden durch den Füllalgorithmus jedoch nur *teilweise* gefüllte Templates geliefert, so wird dem Experten die Möglichkeit gegeben, ein solches in einem Editor zu editieren.

- **Editieren unvollständig gefüllter Templates**
  Hat der Experte ein unvollständig gefülltes Template ausgewählt, so wird dieses in einen Template-Editor kopiert, welcher ein *restriktives* Editieren erlaubt. Restriktiv insofern, als daß nur bestimmte, vom Typ des Templates abhängige Passagen editiert werden können. INFOCOM unterstützt den Experten bei dieser Arbeit durch das Vorschlagen von Alternativen (siehe Kapitel 6.1.1.).

---

[3]Die Organisation der Wissenseinheiten geschieht in COKAM+ in einem sogenannten "Unit-Stack" [Laufkötter91], welcher anhand von Suchkriterien durchlaufen werden kann.

5. Automatisches Füllen abstrakter Templates

Aufgrund einer Wissenseinheit und eines ausgewählten, abstrakten Templates werden automatisch durch INFOCOM (teilweise) gefüllte, formale Wissenseinheiten produziert. Dabei stellen der Constraint-, bzw. der Premise- und Conclusion-Part jeweils die für einen Templatetyp zu füllenden Pattern (Muster) dar. Das automatische Füllen dieser Pattern birgt das Problem in sich, daß die Eigenschaften der Pattern eines abstrakten Templates (premise-, conclusion, bzw. constraint-part) nicht in geeigneter Weise beschrieben werden können.

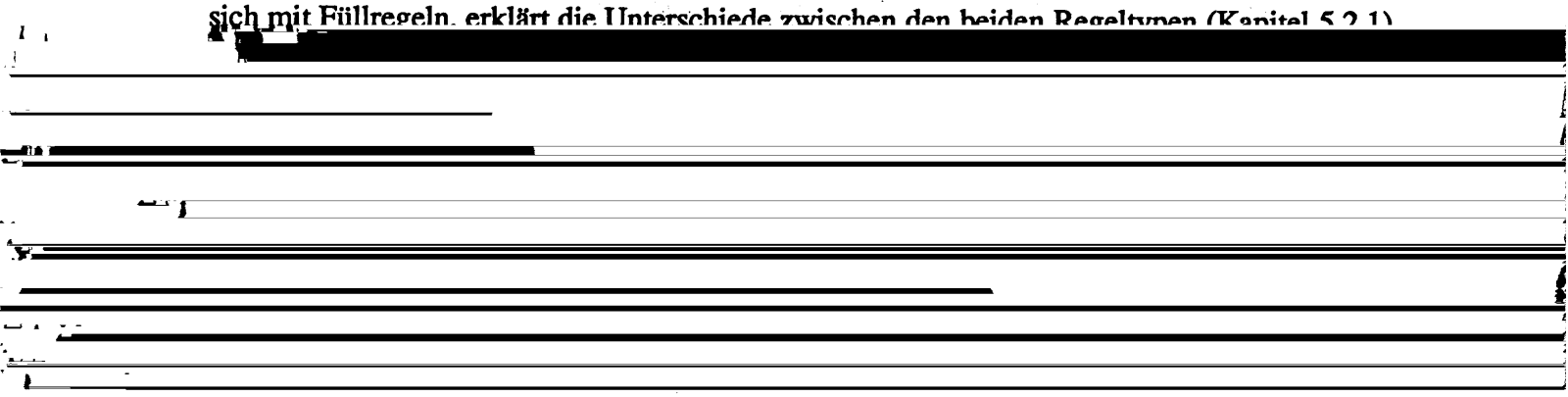
Kapitel 5.1 zeigt, wie das Problem des Füllens eines abstrakten Templates in Teilprobleme zerlegt wird, indem diese Pattern sukzessiv in Unterpattern aufgeteilt werden.

Für die zu füllenden Pattern der in Kapitel 2.1.3 beschriebenen Template-Grundtypen werden in INFOCOM Elementtypen eingeführt; - Unterpattern werden entsprechend durch Sub-Elementtypen beschrieben. Auf einem Elementtyp werden Mengen und Prädikate definiert.

Die oben beschriebenen Pattern und Unterpattern sind die Platzhalter eines abstrakten Templates, die mit Wörtern der Wissenseinheit gefüllt werden sollen. Produzieren von Elementen für diese Platzhalter kommt entsprechend dem Lösen eines jeweiligen Teilproblems gleich, wobei ein erzeugtes Element eine gefundene Lösung hierfür darstellt. Die Elemente (Lösungen) eines Pattern werden in einer hierfür definierten Menge gesammelt.

In INFOCOM erfüllen Prädikate zwei Aufgaben. Zum einen dienen sie zum Testen von Elementen in Bezug auf verschiedene Eigenschaften (Kapitel 5.1.2.). Zum anderen ermöglichen sie das Einbeziehen externer Funktionalitäten in den Füllprozeß.

Der Inhalt von Mengen kann durch sogenannte Füllregeln beeinflußt werden. In Abhängigkeit von ihrem Typ (konstruktiv, destruktiv) fügen Füllregeln Elemente in eine Menge ein oder entfernen sie. Dadurch garantieren sie, daß die Elemente einer Menge nur die für sie relevanten Eigenschaften besitzen (und die für sie unrelevanten nicht). Kapitel 5.2 beschäftigt sich mit Füllregeln, erklärt die Unterschiede zwischen den beiden Regeltypen (Kapitel 5.2.1)



und zeigt, wie Elemente durch konstruktive Füllregeln bewertet werden.

5.1. Aufteilung des Problems des Formalisierens in Teilprobleme

Bevor ein Algorithmus zum Füllen abstrakter Tempates angegeben wird, werden zunächst die verschiedenen Templatetypen näher untersucht. Hierbei gilt es, die zu füllenden Pattern eines Templates in geeigneter Weise in eine Reihe *elementarer* Pattern zu zerlegen

5.1.1. Zerlegung abstrakter Templates

In Kapitel 2.1.3. wurden die drei konkreten Typen abstrakter Templates aufgelistet, die in COKAM+ als Vorgabe beim Formalisieren von Wissen zur Verfügung stehen. Diese lassen sich auf zwei *generische* Grundtypen abstrakter Templates reduzieren:

condition-templates
Durch den generischen Template-Grundtyp *condition-template* werden abstrakte Templates beschrieben, welche eine *Prämisse* und eine *Konklusion*, d.h. eine *Ableitungsrichtung* haben. Die abstrakten Templatetypen precondition- und consequence-template, welche die Vor- und Nachbedingungen eines Operators in der Zieldomäne von COKAM+ beschreiben, sind von diesem Grundtyp. Die allgemeine Form eines condition-templates ist

condition-template ::= template-type (premise -> conclusion)

In dieser Domäne (Maschinenbau) zerfällt die Prämisse in zwei Teile: In einen Operator und eine von diesem abhängige Prämisse. Für die precondition- und consequence-templates ergibt sich daraus:

precondition-template ::= precondition (operator premise -> conclusion)

und

consequence-template ::= consequence (operator premise -> conclusion)

constraint-templates
Im Gegensatz zu condition-templates besitzen *constraint-templates* keine Ableitungsrichtung.

constraint-template ::= template-type (constraint-part)

Die in COKAM+ verwendeten ar-rules sind abstrakte Templates dieses generischen Template-Grundtypen, bei denen der constraint-part analog zu oben in die Teile "operator" und "constraint" zerfällt:

ar-rule ::= ar-rule (operator constraint)

Neben diesen sogenannten *ungewichteten* constraint-templates werden in einer anderen Version von COKAM+ *gewichtete* Templatetypen untersucht. Auf diese soll hier nicht weiter eingegangen werden.

Für diese zwei generischen Templatetypen können die zu füllenden Pattern, der premise-, der conclusion- und der constraint-part, in weitere Unterpattern zerlegt werden. Jedes dieser Pattern besteht aus einem oder mehreren *View-Elementen.* Ein View-Element ist ein mit einem View des Modells der Expertise assoziiertes Wortpaar, welches aus einem *view-namen* und einem *view-value* besteht. Ein solches Wortpaar beschreibt den Zusammenhang zwischen einer Begrifflichkeit (view-name) der Zieldomäne und einem ihm zugeordneten Wert (view-value).

Allgemein läßt sich damit z.B. ein precondition-template mit seinen Unterpattern auf folgende Weise detailliert darstellen:

```
precondition-template   ::= precondition (operator premise-part --> conclusion-part)
                          = precondition (operator  {view-element}*
                                                    -->
                                                    {view-element}*)
                          = precondition (operator  {view-name  (view-value)}*
                                                    -->
                                                    {view-name (view-value)}*)
```

Die nachfolgende Abbildung zeigt ein bereits gefülltes precondition-template (condition-template), bei dem die einzelnen Bestandteile gekennzeichnet sind:

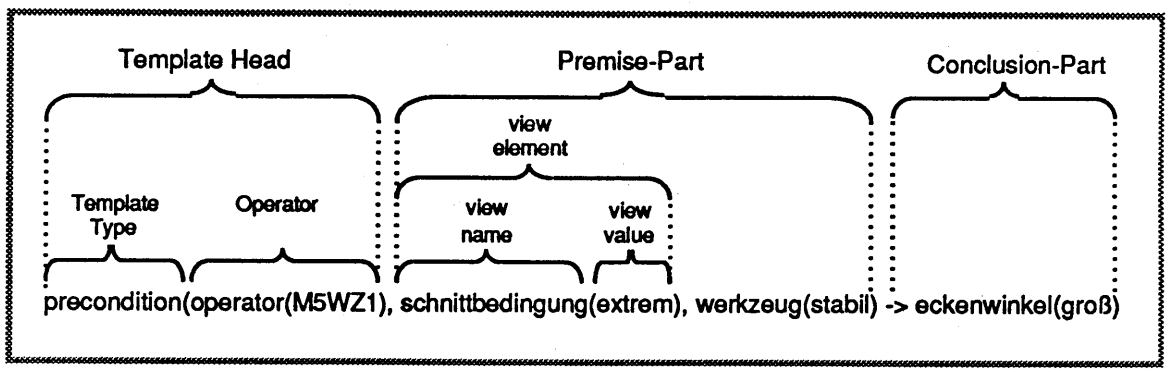


Abbildung 5.1.: Aufteilung eines gefüllten Templates

5.1.2. Aufbau eines Lösungsweges

Im vorhergehenden Kapitel wurde gezeigt, wie sich abstrakte Templates in Pattern zerlegen lassen. Jedem dieser Pattern wird ein *Elementtyp* (string, view-element, ..) zugewiesen, der die Struktur des Pattern beschreibt. In Elementen, die von dem Elementtyp *string* sind, können Wörter, Formeln, Zahlen, usw. aufgenommen werden, die im Text der Wissenseinheit vorkommen ("Schnittbedingung", "Eckenwinkel", "extrem", "45°").

Der Elementtyp view-element beschreibt ein geordnetes Tupel von Elementen des Typs string, bei denen die erste Stelle als *view-name* und die zweite Stelle als *view-value* bezeichnet wird. Elemente dieses Typs stellen einen Zusammenhang zwischen einem Begriff und einem Wert dar ("Schnittbedingung extrem", "Eckenwinkel 45°").

Um Elemente eines Typs auf bestimmte Eigenschaften hin testen zu können, werden auf den Elementtypen *Prädikate* definiert, die im folgenden behandelt werden sollen.

Prädikate

Durch die Zerlegung eines abstrakten Templates lassen sich die Eigenschaften angeben, welche ein Element aufweisen muß, das für ein Pattern vorgeschlagen werden soll. Beispielsweise sind

i.) "view-names sind überwiegend DIN-Zeichen oder Nomen."

ii.) "view-values sind am häufigsten Zahlen, Formeln oder Adjektive."

Aussagen, welche Eigenschaften der Pattern view-name und view-value anzeigen.
Das Testen eines Elements auf eine bestimmte Eigenschaft wird durch *Prädikate* ermöglicht. Für einen Elementtypen werden diese jeweils *direkt* in der Programmiersprache LISP eingegeben, so daß die in der LISP-Umgebung vorhandenen Funktionen genutzt werden können. Damit läßt sich beispielsweise die Eigenschaft

"stehen view-name und view-value im selben Kasus"

durch

```
casus-equal-p (view-element)

<=>
```

```
#'(lambda (element)
   (let   ((view-name     (first element))
           (view-value    (second element)))
     (equal
          (get-kasus view-name)
          (get-kasus view-value))) view-element)
```

liefert den Wert T (true)

testen, falls die entsprechende Funktionalität einer Morphologie in der LISP-Ungebung vorhanden ist. Die oben gezeigte LET-Anweisung spiegelt die Struktur des Elementtypen view-element wieder. Diese wird in INFOCOM in einem Prädikat-Editor automatisch generiert, so daß hier nur noch der Rumpf der Lambda-Funktion eingegeben werden muß.

Mengen

Während des Füllens eines abstrakten Templates durchläuft der Füllalgorithmus einen *gerichteten Graphen*. An den *Knoten* dieses Graphen sind *Mengen* angehängt, in denen die während des Füllvorgangs gefundenen Teillösungen, d.h. die für ein Pattern vorgeschlagenen Elemente, gesammelt werden. Diese Mengen sind mit einer Anzahl von anderen, sogenannten *Quellmengen*, assoziiert. Hat eine Menge keine Quellmengen, so wird diese als *Grundmenge* bezeichnet (z.B. die Menge unit-words, welche die Wörter der Wissenseinheit beinhaltet). Grundmengen sind spezielle Quellmengen und werden vor dem eigentlichen Füllvorgang mit den für sie vorgesehenen Elementen gefüllt.

Diese Assoziationen von Mengen und Quellmengen bilden die gerichteten *Kanten* des Graphen. Sie kennzeichnen den Weg, auf dem sich die Elemente von den Grundmengen bis zu den Platzhaltern eines abstrakten Templates bewegen. Diese Kanten können mit *Eigenschaften* belegt werden, welche eine "Ventil-Funktion" bilden. So werden auf einer Kante nur Elemente "befördert", welche die mit der Kante assoziierten Eigenschaften besitzen.

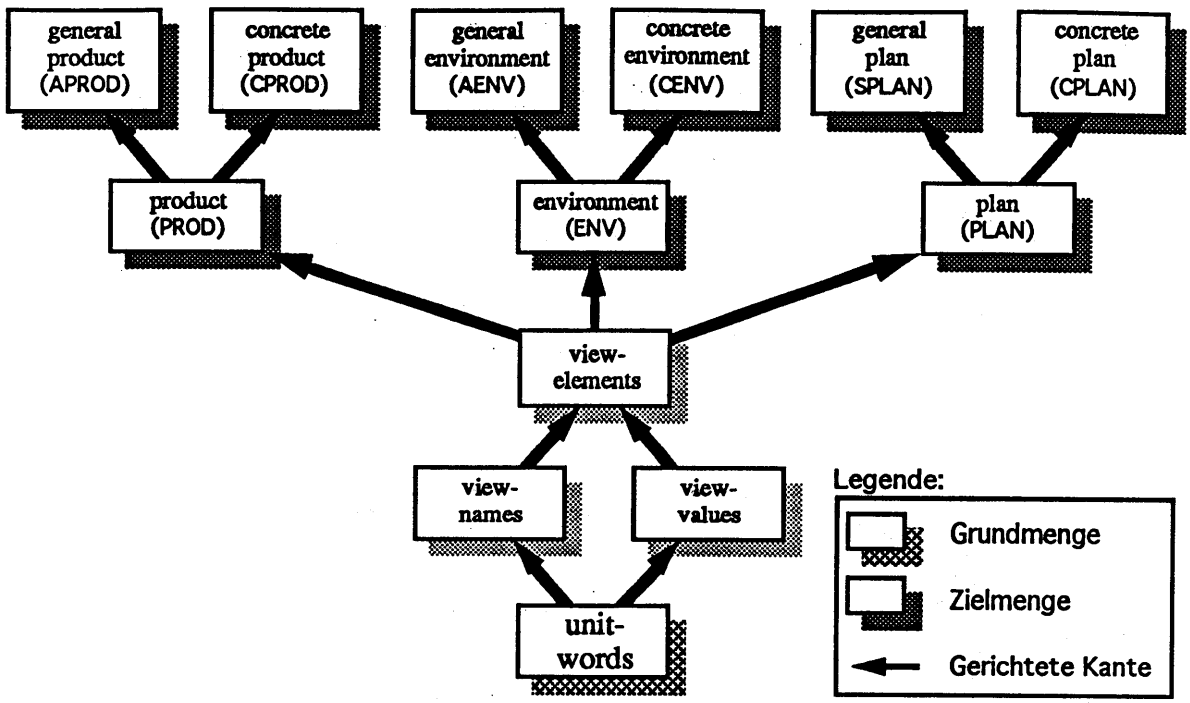


Abbildung 5.2: Der gereichtete Graph des Füllalgorithmus.
Ausgehend von der Grundmenge "unit-words" durchlaufen die Elemente den Graphen bis zu den Zielmengen ("general product",..., "concrete plan").

Auf den Knoten und Kanten, die zwischen den Grundmengen und den Zielmengen liegen, realisieren *Füllregeln* (siehe Kapitel 5.2.) die oben genannten "Ventil-Funktionen", welche für die Elemente das Vorhandensein der gewünschten Eigenschaften garantieren.

Die folgende Abbildung zeigt ein abstraktes Precondition-Template und den Graphen aus Abb. 5.2. nach durchlaufen des Füllalgorithmus. Die Zielmengen des Graphen sind durch die in dem Template auftretenden Views *environment*, *plan* und *concrete plan* des Modells der Expertise festgelegt (Symbole *ENV*, *PLAN* und *CPLAN*).

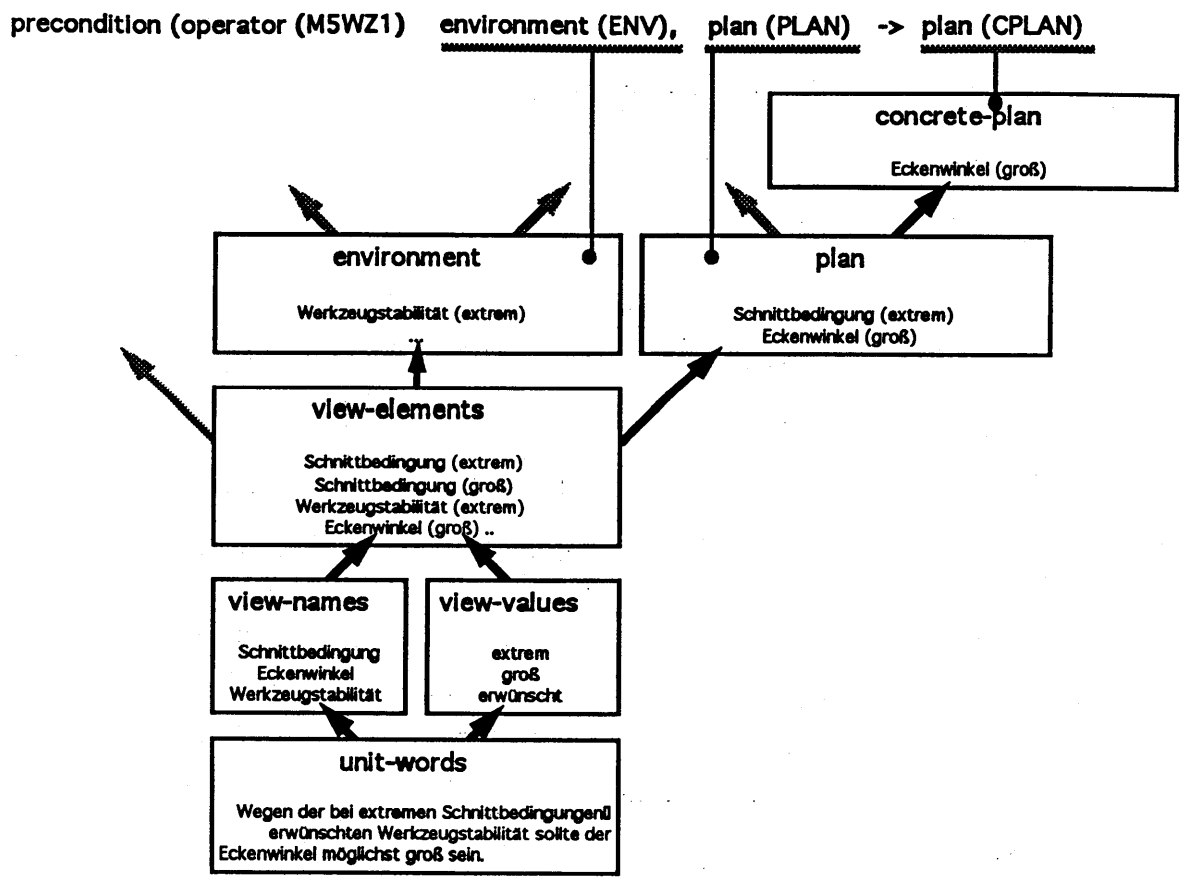


Abbildung 5.3.: Der gerichtete Graph nach durchlaufen des Füllalgorithmus.
In den Knoten sind jeweils die für eine Menge gefundenen Kandidaten aufgelistet.

Aus der Menge aller Wörter der Wissenseinheit (unit-words) werden Kanditaten für die Mengen view-names und view-values bestimmt (insbesondere Nomen bzw. Adjektive). Die Menge view-elements enthält Paare von Wörtern aus ihren beiden Quellmengen view-names und view-values (Überprüfung der zwei Wörter auf gleichen Kasus, Numerus, usw.). environment, plan und concrete plan stellen die Zielmengen für das zu füllende, abstrakte Template dar und enthalten Kandidaten, die für die entsprechenden Platzhalter vorgeschlagen werden sollen.

5.2. Steuern des Füllalgorithmus durch Füllregeln

Im folgenden werden Mechanismen eingeführt, mit denen Operationen auf Mengen durchgeführt werden können. Diese Mechanismen werden durch Füllregeln[4] realisiert, die auf einer Menge definiert werden. Füllregeln werden in drei Bereiche aufgeteilt:

- Der **Quantor-Part** legt den Definitionsbereich einer Füllregel fest. Er liefert (sortengerecht) die für die Regel benötigten Argumente.
- In der **Prämisse** werden die Argumente des Quantor-Parts in geeigneter Weise durch logische Operatoren und Prädikate zu einer Bedingung verknüpft.
- In der **Konklusion** wird eine Aktion beschrieben, die bei Zutreffen einer Füllregel ausgeführt werden soll. Trifft die Bedingung der Prämisse auf die vom Quantorpart gelieferten Argumente zu, so wird die durch die Konklusion beschriebene Aktion ausgeführt, d.h. die Regel *feuert.*

Eine formale Beschreibung einer Füllregel kann nun wie folgt angegeben werden:
Seien M, $M_1,\ldots, M_n$ Mengen, $m_1,\ldots, m_n$ Elemente mit $m_i \in M_i$. Sei $\Re$ die Menge der Füllregeln.

Eine **Füllregel** $\rho \in \Re$ auf M ist eine Abblidung $\rho: M_1 \times \ldots \times M_n \rightarrow M$ mit

$$\rho(m_1,\ldots, m_n) = \underline{\text{if}}\ \text{Premise}(\rho, m_1,\ldots, m_n)\ \underline{\text{then}}\ \text{Conclusion}(\rho)$$

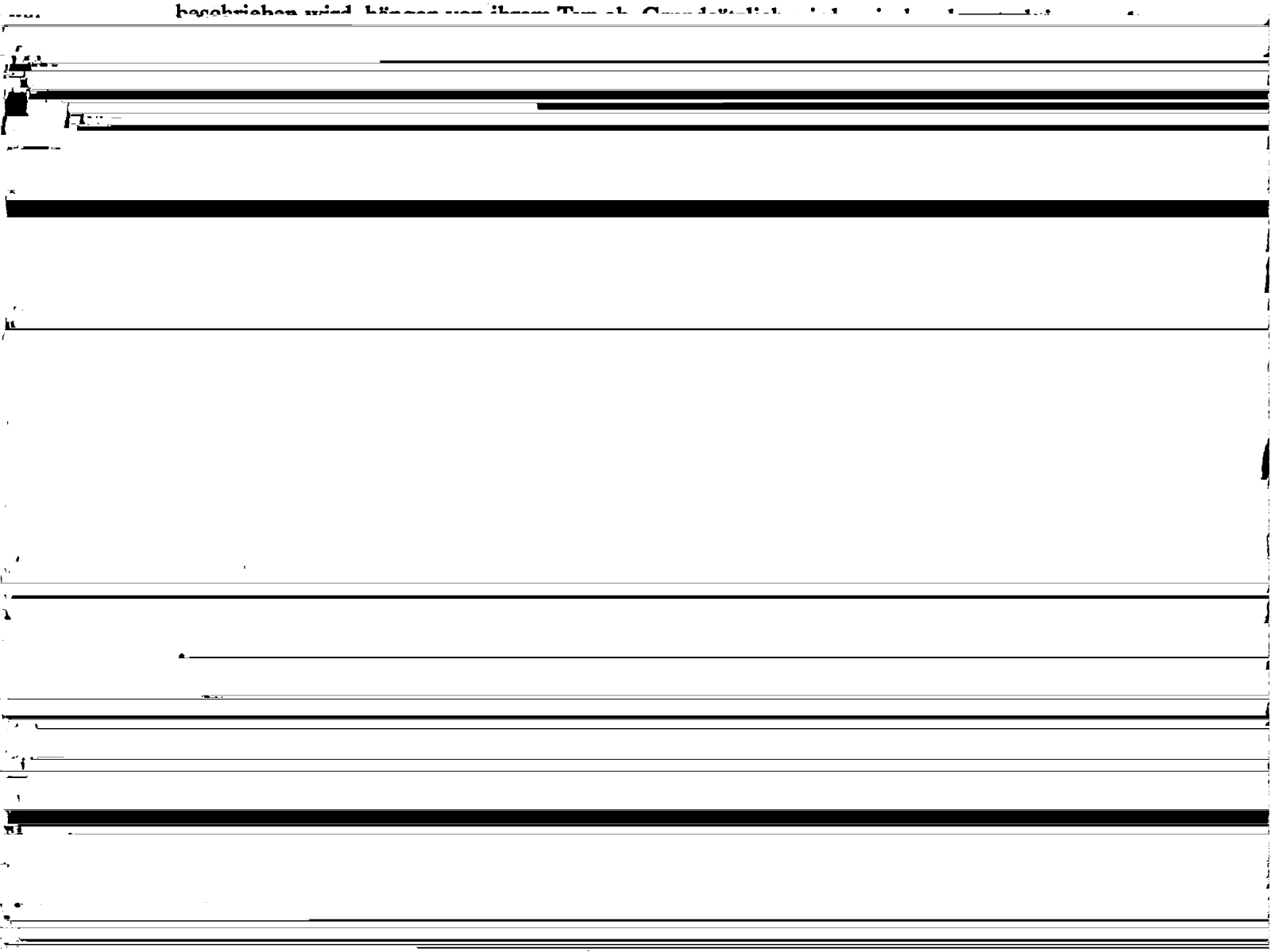
Hierbei ist Premise: $\Re \times M_1 \times \ldots \times M_n \rightarrow$ (true, false) ein Algorithmus mit

$$\text{Premise}(\rho, m_1,\ldots, m_n) = \begin{cases} \text{true, falls } m_1,\ldots, m_n \text{ die Prämisse von } \rho \text{ erfüllen,} \\ \text{false, sonst} \end{cases}$$

[4] Regeln sind hier Vorwärtsregeln. Trifft die Prämisse auf die vom Quantorpart gelieferten Argumente zu, so wird die in der Konklusion beschriebene Aktion ausgeführt.

5.2.1. Regeltypen

Sowohl der Definitionsbereich einer Füllregel, als auch die Aktion, die durch eine Füllregel beschrieben wird, hängen von ihrem Typ ab. Grundsätzlich [illegible]



destruktiven Füllregeln unterschieden.

Konstruktive Füllregeln :

Konstruktive Füllregeln iterieren auf den Quellmengen der Menge, auf denen sie definiert sind. Sie werden wie folgt beschrieben:
Sei M eine Menge, $M_1,.., M_n$ die Quellmengen von M. Sei Assessment:$\Re$ -> [0, 100] eine Funktion mit Assessment (ρ) = a, wobei a $\in$ [0,100] eine Bewertung der Füllregeln $\rho \in \Re$ ist. Sei Premise wie oben. Dann gilt:
Eine **konstruktive Füllregeln** ρ_c auf M ist eine Abbildung ρ_c:$M_1 \times ... \times M_n$ -> M mit

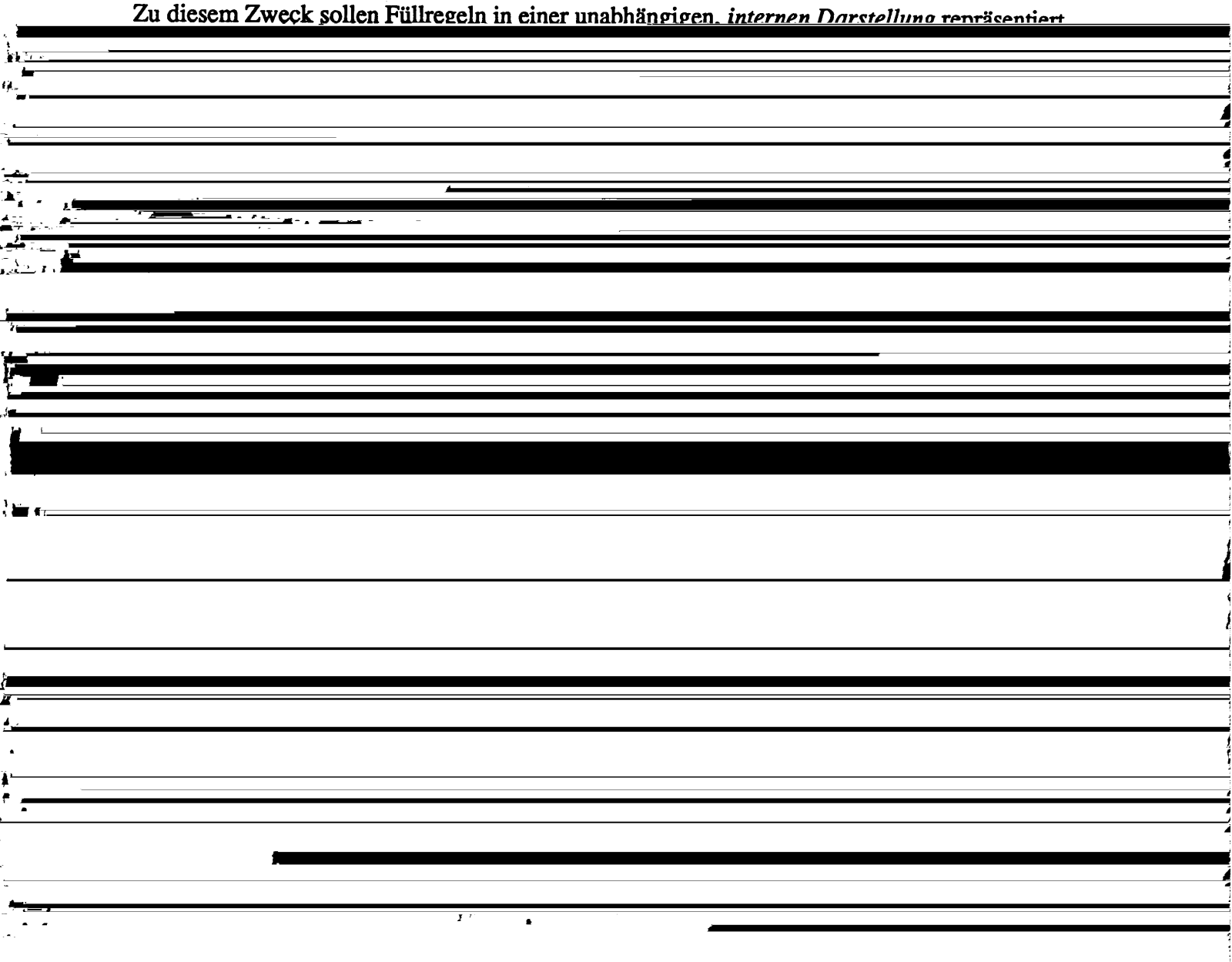
$\rho_c(m_1,..., m_n)$ = if Premise(ρ_c, m_1,..., m_n)
then InsertElement(M,m_1,...,m_n,Assessment(ρ_c))

Konstruktive Füllregeln füllen die Menge auf der sie definiert sind mit den Elementen der Quellmengen, falls diese die in der Prämisse beschriebenen Eigenschaften besitzen

5.2.2. Views einer Füllregel

Auf die gleiche Weise, wie der Experte bei der Formalisierung von Textwissen durch INFOCOM unterstützt werden soll, muß auch der KE durch den Experten unterstützt werden. Dies gilt insbesondere für die Eingabe und das Editieren von Füllregeln. Damit der Experte dem KE hierbei Unterstützung leisten kann, muß dieser die Füllregeln *verstehen* können.

Zu diesem Zweck sollen Füllregeln in einer unabhängigen, *internen Darstellung* repräsentiert



[AlexanderFreiling+87][5], gewonnen werden können. Im folgenden sollen diese vier Views und die daraus resultierenden Anforderungen an die interne Darstellung einer Füllregel erläutert werden:

- **Semantic-View**
  Der Experte muß die Bedeutung der von dem Knowledge-Engineer eingegebenen Füllregeln verstehen, um diesem Verbesserungsvorschläge machen zu können (Verifikation durch den Experten). Hierfür soll die interne Darstellung einer Regel in *natürliche* Sprache überführt werden können. Der *semantic-view* zeigt die Bedeutung einer Füllregel in natürlicher Sprache, damit diese auch von "Nicht-

- **Abstraktion-View**
  Der *abstraktion-view* zeigt eine Füllregel auf der Datenebene als compilierte Funktion. Dieser View wurde nur der Vollständigkeit halber aufgenommen ("<Anonymous Function #x3...> !").

Die nachfolgende Abbildung zeigt die vier Views einer Füllregel und den ihnen zugeordneten Abstraktionsebenen:

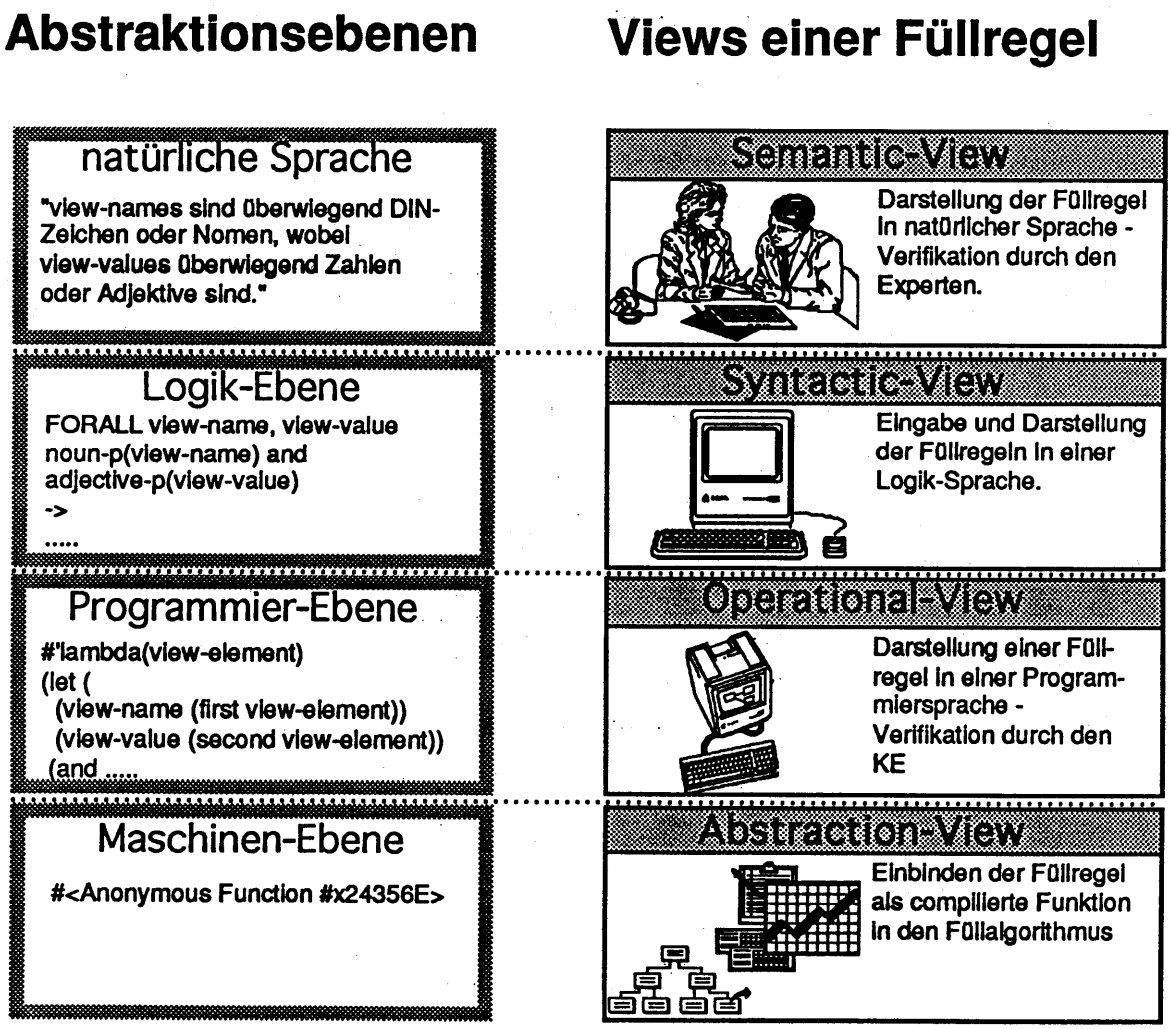


Abbildung 5.3.: Überblick der vier Views einer Füllregel.

Die vier Views ermöglichen demnach das *Verstehen* Außenstehender, das *Eingeben*, das *Verifizieren* und das *Ausführen* einer Regel.

5.2.3. Generierung von Views

Im folgenden sollen Mechanismen besprochen werden, mit deren Hilfe automatisch die oben genannten Views einer Füllregel generiert werden. Dazu wird zunächst die Darstellung angegeben, die INFOCOM für die Verwaltung der Regeln verwendet.

Die *interne Darstellung* einer Füllregel beinhaltet deren Typ und die Prämisse. Aus dem Typ können der Definitionsbereich und die Konklusion bestimmt werden (vgl. 5.2.1.). Die Prämisse besteht aus logischen Operatoren. Diese erlauben es, Prädikate logisch miteinander zu verknüpfen. Hierbei werden die in einer Liste des Operators stehenden Prädikate getestet und die Ergebnisse durch eine mit dem logischen Operator assoziierten Funktion ausgewertet. Sie können durch den KE selbst definiert werden. Für die Definition eines logischen Operators müssen die folgenden Angaben gemacht werden:

Name und logisches Zeichen
Im Syntactic View kann die logische Repräsentation eines Operators als Zeichen, oder als Wort erfolgen. Dafür wird einem Operator ein logisches Zeichen und ein Name zugeordnet. (zum Beispiel: ∨, OR).

Stelligkeit
Für jeden logischen Operator wird dessen *Stelligkeit* angegeben. Dadurch werden Fehler während der Eingabe einer Füllregel vermieden (zweistellig: 2, oder beliebig: *).

[redacted]

Um die Semantik einer Füllregel in natürlicher Sprache wiedergeben zu können, wird einem logischen Operator für jede zur Verfügung stehenden Sprache[6] (Englisch, Deutsch, ..) eine natürlichsprachliche Bedeutungen zugeordnet. ("or", "oder", ...). Für jedes Prädikat, das mit einem logischen Operator assoziiert ist, wird eine natürlichsprachliche Bedeutung ausgewählt und der Bedeutung des Prädikates vorangestellt. Die natürlichsprachliche Bedeutung eines Prädikates wird hierbei in Abhängigkeit seiner Argumentliste angegeben:

Beispiel: Prädikat kasus-equal-p (word1 word2)

Deutsch "stehen word1 und word2 in dem selben Kasus"

Bei der Generierung werden die Platzhalter durch die Argumente der Eingabe ersetzt.

logische Funktion

Jedem Operator wird eine logische Funktion zugeordnet, welche die von den Prädikaten gelieferten Werte als Argumente erhält (die LISP-Funktion OR).

Die logischen Verknüpfungen AND, OR und NOT sind bereits vordefiniert.

Für die automatische Erzeugung der Views werden drei verschiedene Generatoren benötigt:

Semantic View	Syntactic View	Operational View
Diese Regel füllt die Menge View-Elements. Als Argumente dienen die Elemente view-name und view-value und ist view- name ein Nomen und view-value ein Adjektiv dann wird aus den Argumenten ein neues Element erzeugt und mit dem Wert 70 bewertet.	FORALL view-names FORALL view-values AND noun-p view-name adjective-p view-value InsertElement view-element 70	(lambda(element) (let((view-name (first element)) (view-value (second element))) (if (and (noun-p view-name) (adjective-p view-value)) (insert-element element assessment))))

5.3. Regelsysteme

Beim Füllen eines abstrakten Templates kann es passieren, daß entweder kein, oder zuviele gefüllte Templates vorgeschlagen werden. Um diesem Fall entgegen zu treten, können in INFOCOM für die Steuerung des Füllalgorithmus verschiedene *Regelsysteme* eingegeben werden.

Dies ermöglicht zum einen das Zusammenstellen sogenannter "*schwacher*" und "*starker*" Regelsysteme, welche nur Füllregeln schwacher, bzw. starker Ausdruckskraft beinhalten. Konnten aufgrund eines Regelsystems für ein abstraktes Template keine Vorschläge gemacht werden, so wählt man für einen erneuten Füllvorgang ein schwächers Regelsystem aus. Wurde dagegen eine zu große Menge gefüllter Templates vorgeschlagen, so läßt sich diese durch die Wahl eines stärkeren Regelsystems einschränken.

Zum anderen wird dem Experten durch das Eingeben neuer Regelsysteme die Möglichkeit geboten, sich in das Definieren von Füllregeln einzuarbeiten um selbst Regelsysteme erstellen zu können, d.h. den Füllalgorithmus selbst zu optimieren.

5.4. Der Füllalgorithmus

Sei τ ein abstraktes Template mit den Grundmengen $S_1,...,S_p$ und den Zielmengen $D_1,...,D_q$. Sei $R \subset \Re$ ein Regelsystem. Der Füllalgorithmus für τ und R wird wie folgt beschrieben:

A. Initialisiere rekursiv alle Mengen, beginne bei den Zielmengen $D_1,...,D_q$.

B. Fülle die Grundmengen $S_1,...,S_p$ mit den für sie vorgesehenen Elementen.

C. Rufe den Algorithmus Fill-Set mit der Zielmenge D_i auf $(i=1,...,q)$, der wie folgt

realisiert ist.

Fill-Set(M)

1. besimme alle Quellmengen $Q_1,...,Q_n$ von M
2. IF n = 0 THEN STOP
3. rufe den Algorithmus Fill-Set mit jeder Quellmenge Q_i auf.
4. bestimme alle Bewertungsregeln $\rho_{c_1},...,\rho_{c_r}$ von M aus R
5. bestimme alle Filtrationsregeln $\rho_{d_1},...,\rho_{d_s}$ von M aus R
6. bestimme die Menge $M_C = \{m \mid m = (m_1,...,m_n)$ mit $m_i \in M_i\}$ aller geordneter n-Tupel der Mengen Q_i, i = 1,...,n.
7. FORALL $m_i \in M_C$
8. bestimme aus den $\{\rho_{c_1},...,\rho_{c_r}\}$ die *beste* Bewertungsregel ρ_c, die auf m_i zutrifft, d.h. die Bewertungregel mit der höchsten Bewertung.

9. bestimme eine Filtrationsregel ρ_d, die auf m_i zutrifft.
10. IF ρ_c and $\neg \rho_d$ THEN
11. Insert-Element (M, m_i, (assessment (ρ_c)))

D. Setze die gefundenen Lösungen, d.h. alle Elemente der Zielmengen $D_1,...,D_q$, zu Templates zusammen und stelle sie in dem Hauptdialog dar.

Durch den Algorithmus Fill-Set wird als erstes, ausgehend von den Zielmengen $D_1,...,D_q$, der für den Typ des abstrakten Templates τ zugeordnete Graph durchlaufen (3), bis eine Grundmenge erreicht wird (1). Ist die erreichte Menge (M) keine Grundmenge, so werden die für sie eingegebenen Bewertungs- (4) und Filtrationsregeln (5) aus dem Regelsystem R bestimmt.

Anschließend wird über der Menge M_c iteriert, welche alle geordneten n-Tupel m_{ck} = $(m_1,...,m_n)$ mit $m_i \in Q_i$ enthält, wobei $|M_c| = |Q_1| * ... * |Q_n|$ gilt.

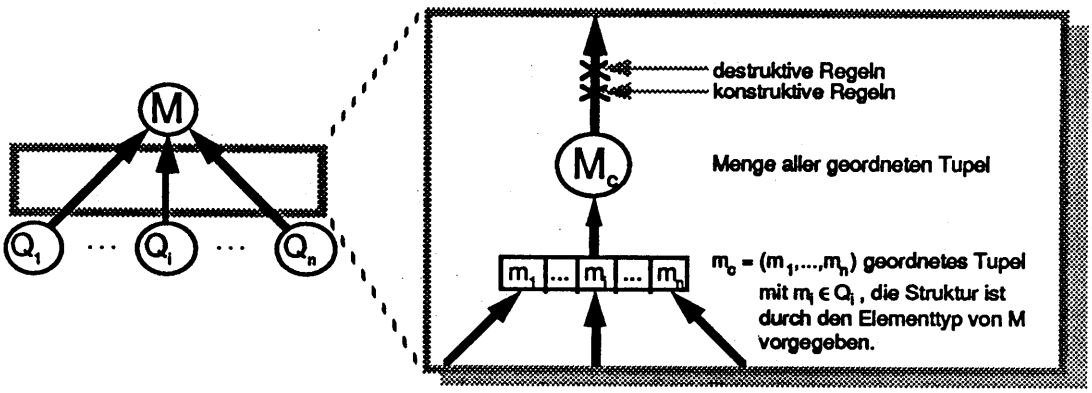


Abbildung 5.4.: Eine Menge mit ihren n Quellmengen.
Die Elemente der Quellmengen werden zu geordneten n-Tupeln kombiniert und in der Menge M_c abgelegt.

Für jedes der Elemente $m_{ck} \in M_c$ wird *die* Bewertungsregel mit der höchsten Bewertung gesucht (8). Falls eine solche Bewertungsregel gefunden wird und **keine** Filtrationsregel auf das Element m_{ck} zutrifft (10), so wird aus m_{ck} ein neues Element m erzeugt, mit einer Bewertung versehen und in die Menge M aufgenommen (11). Diese Bewertung ist der Mittelwert der für die Elemente $m_1,...,m_n$ bisher bestimmten Bewertungen und der Bewertung der Bewertungsregel ρ_c, die auf das Element m zutrifft:

$$\text{assessment (m)} = (\text{assessment } (\rho_c) + \text{assessment } (m_1) + ... + \text{assessment } (m_n)) / (n + 1)$$

Im folgenden Beispiel soll gezeigt werden, wie während des Füllvorgangs Kandidaten für eine Menge durch Füllregeln vorgeschlagen und bewertet werden.

Zu füllende Menge M: view-elements

Quellmengen Q_1 und Q_2: view-names und view-values

Bewertungsregel ρ_c:

```
FORALL view-names
FORALL view-values
      AND
            noun-p (view-name)
            adjective-p (view-value)
->
InsertElement ((view-name view-value) 80)
```

Filtrationsregel ρ_d:

```
FORALL view-elements
      NOT
            equal-gender-p (view-element)
->
RemoveElement (view-element)
```

1.) Für die Belegung view-name = ("Schnittbedingungen".50)[7] und view-value = ("schnelles".50) wird Premise(ρ_c,view-name,view-value) wahr und es wird ein neues View-Element (("Schnittbedingungen" "schnelles").60) erzeugt. Da die beiden Wörter jedoch nicht dasselbe Geschlecht haben, wird das neue Element nicht in die Menge view-elements eingefügt (Premise(ρ_d,view-element) = true).

2.) Für view-name = ("Schnittbedingungen".50) und view-value = ("extreme" 50) ist Premise(ρ_c,view-name,view-value) immer noch wahr. Da jetzt die Filtrationsregel ρ_d nicht mehr auf das neu erzeugte Element zutrifft, wird ein neues n-Tupel (vgl. Abbildung 5.4, wobei n=2) mit der Bewertung

$$\frac{\text{assessment}(\rho_c) + \text{assessment(view-name)} + \text{assessment (view-value)}}{3} = \frac{80 + 50 + 50}{3} = 60$$

in die Menge view-elements eingefügt.

[7] Jedes Element einer Menge ist eine CONS-Liste, bestehend aus dem Inhalt (car-Teil) und der Bewertung (cdr-Teil).

6. Arbeiten mit INFOCOM

Das Arbeiten mit INFOCOM ist in zwei getrennte Modi unterteilt, den Expert- und den KE-Mode. Dem Experten bzw. dem Knowledge-Engineer werden in diesen Modi verschiedene Dialogfenster und Editoren geboten, anhand derer das Arbeiten und die Funktionsweise von INFOCOM kurz beschrieben werden soll.

6.1. Der Expert-Mode

Der Expert-Mode befaßt sich ausschließlich mit den (oben genannten) drei Phasen des Formalisierens von Textwissen. Mittelpunkt bildet hierbei der Hauptdialog zum Füllen abstrakter Templates:

6.1.1. Füllen abstrakter Templates

Beim Formalisieren einer Wissenseinheit werden von COKAM+ abstrakte Templates generiert. Diese Formalisierungsvorgaben werden in INFOCOM zusammen mit dem Inhalt der Wissenseinheit in einem Dialog dargestellt (vgl. Kapitel 4.1.). Der Experte bestimmt mit Hilfe seiner Expertise ein der Wissenseinheit entsprechendes, abstraktes Template, indem er durch das Anklicken der Buttons "previous" und "next" die (ungeordnete) Liste der vorgeschlagenen Templates durchsucht.

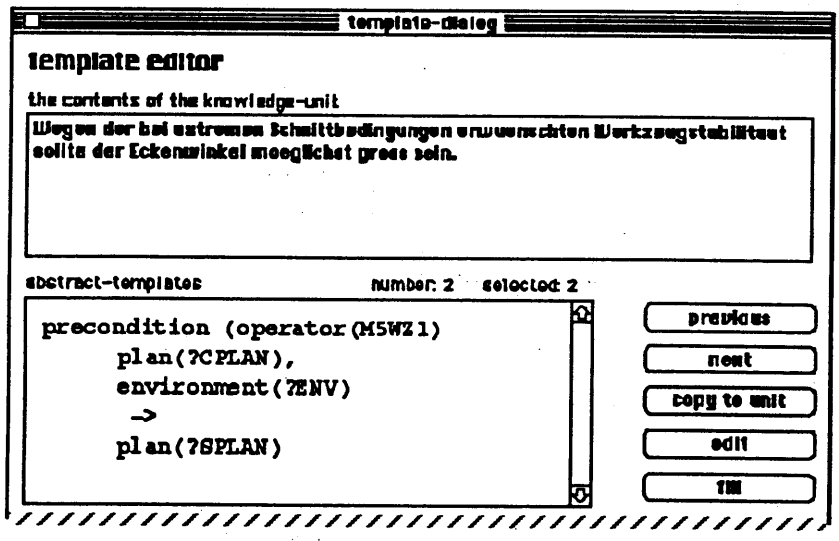


Abbildung 6.1.: Oberer Teil des Hauptdialogs mit
dem Inhalt der Wissenseinheit und dem Fenster zum selektieren der abstrakten Templates.

In den abstrakten Templates sind die zu füllenden, bzw. die zu editierenden Passagen durch die Ein-/ Ausgabeklassen des Modells der Expertise gekennzeichnet (CPLAN für *concrete*

product, ENV für *environment*, SPLAN für *general plan*, siehe [Huf94, in press]). Hat der

füllen lassen (Button "fill").

Die aufgrund des selektierten Regelsystems durch den Füllalgorithmus gefundenen Vorschläge gefüllter Templates werden in einem weiteren Auswahlfenster in Abhängigkeit ihrer Bewertung *absteigend sortiert* aufgelistet. Da das am höchsten bewertete, gefüllte Template *leider* nicht immer das gewünschte ist, d.h. nur teilweise- oder unkorrekt gefüllte Passagen enthalten kann, muß auch hier eine Auswahl getroffen werden.

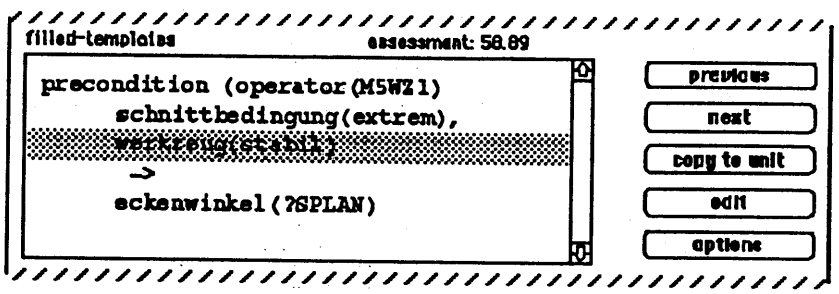


Abbildung 6.2.: Mittlerer Teil des Hauptdialogs

Im Template-Editor wählt der Experte ein View-Element aus, das durch den Füllalgorithmus nicht korrekt gefüllt worden ist. Hierzu selektiert er dieses mit der Maus und kopiert es mit Hilfe des Buttons "edit" in den Line-Editor.

Beim Korrigieren des ausgewählten View-Elements wird der Experte anhand von Alternativvorschlägen unterstützt, die durch den Füllalgorithmus für die Mengen view-names, view-values und view-elements gefunden wurden.

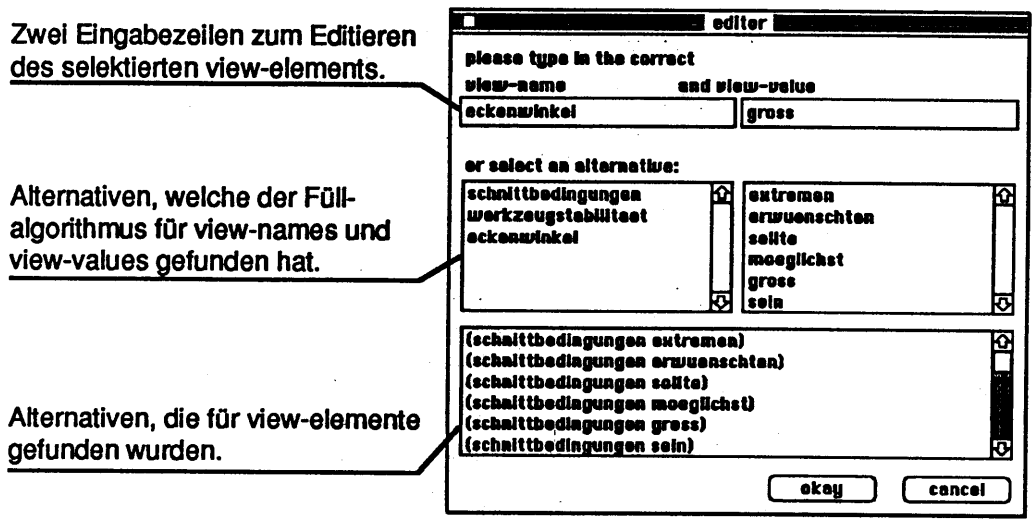


Abbildung 6.4.: Der Line-Editor mit den zwei Eingabezeilen. Die Elemente der Mengen view-names, view-values und view-elements werden als Alternativen für das zu editierende View-Element vorgeschlagen.

Diese Vorschläge können an die entsprechenden Stellen (view-name, view-value) des zu korrigierenden View-Elements kopiert und dort weiter editiert werden.

6.1.2. Optionen des Füllalgorithmus

Um die Ergebnisse des Füllalgorithmus zu beeinflussen, kann der Experte durch Anklicken des Buttons "options" verschiedene Optionen auswählen. Es wird folgendes Dialogfenster dargestellt:

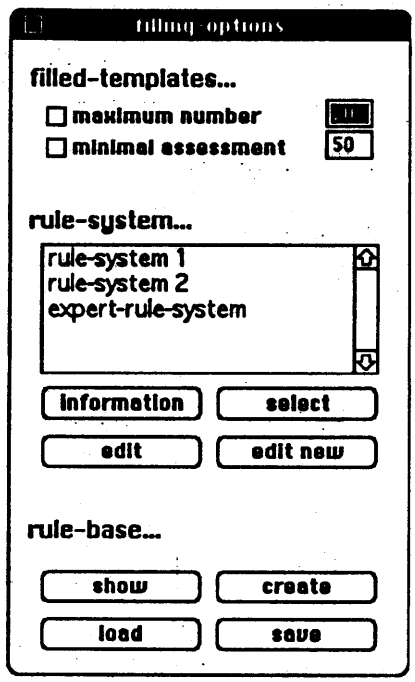


Abbildung 6.5.: Dialogfenster mit Fülloptionen.

Wechseln des Regelsystems

Beim Füllen eines abstrakten Templates kann es passieren, daß entweder kein, oder zuviele gefüllte Templates vorgeschlagen werden. Aufgrund einer Information, die von dem Knowledge-Engineer für jedes Regelsystem eingegeben wurde, kann der Experte im Option-Dialog ein anderes System auswählen und den Füllvorgang erneut starten.

Einschränken der gelieferten Ergebnisse

Um die vom Füllalgorithmus gelieferte Anzahl der Ergebnisse auf eine andere Weise einzuschränken, kann der Experte zwei Werte eingeben.

Der eine setzt die maximale Anzahl der darzustellenden Vorschläge fest, während der zweite Wert einen Schwellwert für die Bewertung bestimmt, die ein Vorschlag mindestens haben muß.

Die Abbildung 6.5 zeigt den erweiterten Optionen-Dialog im sogenannten Knowledge-Engineer-Mode, in welchem der Zugriff auf die Regel-Basis erlaubt wird. Das nachfolgende Kapitel soll eine Beschreibung dieses Benutzermodus geben.

6.2. Der Knowledge-Engineer-Mode

Der Knowledge-Engineer-Mode befaßt sich überwiegend mit der Konfigurierung des Füllalgorithmus. Diese Arbeit ist zum einen durch das Eingeben von Regeln gekennzeichnet, welche zusammengefaßt zu Regelsystemen den Füllalgorithmus steuern. Zum anderen wird dem Knowledge-Engineer die Möglichkeit gegeben, neue Prädikate zu definieren. Die für diese Arbeit bereitstehenden Hilfsmittel sollen im folgenden dargestellt werden.

6.2.1. Darstellung der Regelbasis durch Grapher-Windows

Alle für den Füllalgorithmus notwendigen Objekte werden in einer *Regelbasis* zusammengefaßt. Diese Objekte können in verschiedenen Grapher-Windows[8] mit jeweils den mit ihnen assoziierten Objekten als Baum dargestellt werden. In INFOCOM werden für die Darstellung der Objekte drei verschiedene Knotentypen verwendet:

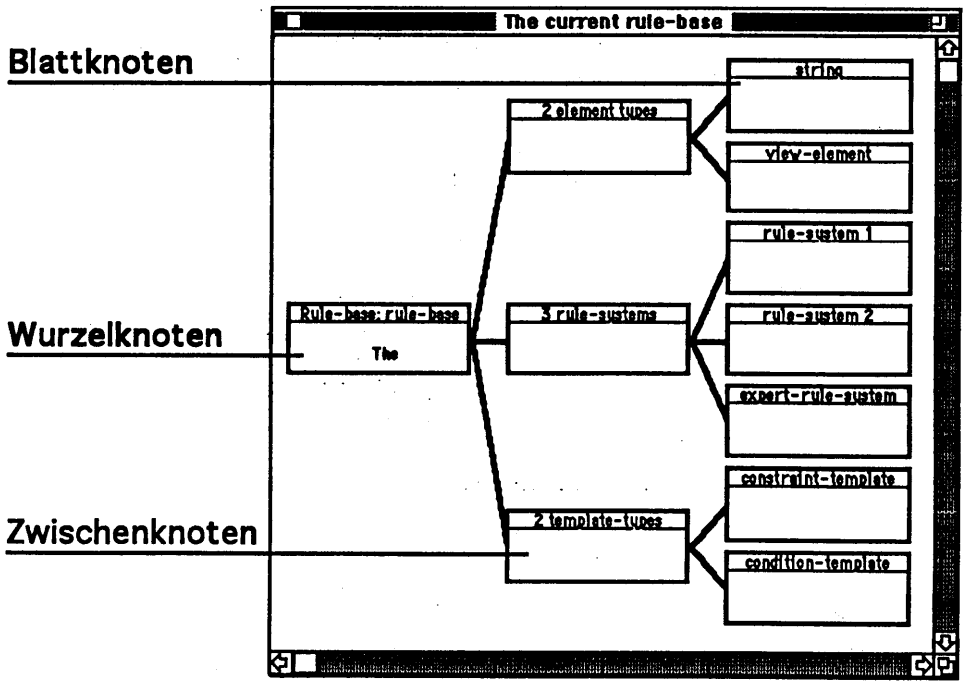


Abbildung 6.5.: Das Grapher-Window der Regelbasis
mit den drei verschiedenen Knotentypen

[8] Grapher V. 2.0, G. Waloszek, Universität Trier, angepaßt an Allegro CL 1.3.1. Lisp von O. Kühn und G. Laufkötter , DFKI Kaiserslautern. Der Listen-Grapher ermöglicht die Darstellung von Listen und Objekt-Strukturen sowie den interaktiven Dialog mit dem Benutzer durch das Reagieren auf Key- und Mouse-Events.

Neben den verschiedenen Bedeutungen dieser Knotentypen sind auf ihnen verschiedene Aktionen definiert, die beim Selektieren eines Knotens mit der Maustaste durchgeführt werden:

- **Wurzelknoten**
  Jedes Grapher-Window enthält genau einen Wurzelknoten, der das darzustellende Objekt repräsentiert. Durch einen einfachen Mausklick auf einen Wurzelknoten wird ein Fenster geöffnet, welches Informationen über das selektierte Objekt enthält (sofern diese durch den KE eingegeben worden sind).

- **Zwischenknoten**
  Zwischenknoten repräsentieren die Ausprägungen des darzustellenden Objekts. Durch einen einfachen Mausklick auf einen Zwischknoten werden die Objekte, welche mit dem Objekt des Wurzelknotens über die Ausprägung assoziiert sind, ein- bzw. ausgeblendet, was das Durchsuchen der Elemente der Regelbasis erleichtert. Soll ein neues Element für eine Ausprägung eingefügt werden, so wird durch einen "Option-Klick" auf einen Zwischenknoten ein entsprechender Editor (siehe nächstes Kapitel) aufgerufen.

- **Blattknoten**
  Die Blattknoten stellen die mit dem Objekt des Wurzelknotens assoziierten Objekte dar. Mit einem "Option-Click" wird hier ebenfalls ein entsprechender Editor aufgerufen, s. d. das ausgewählte Objekt editiert werden kann. Das Entfernen eines Objektes aus dem Baum erfolgt durch einen "Command-Option-Click" (Maustaste mit gedrückter Command- und Option-Taste).

Diese Darstellungsform gewährleistet einen guten Überblick über die Objekte der Regelbasis und ermöglicht den Zugang zu verschiedenen Editoren, die im folgenden behandelt werden.

6.2.2. Regelsysteme und Regeln

Zu den Optionen des Füllalgorithmus (Kapitel 6.1.2.) zählt die Möglichkeit, ein Regelsystem aus einer Menge verschiedener Systeme auszuwählen. Hierfür gibt der Knowledge-Engineer bei der Definition eines neuen Regelsystems den Namen und eine kurze Beschreibung an, in welcher der Experte einen Überblick der vorhandenen Füllregeln erhält. Nach einem "Option-Click" auf einen Zwischenknoten in dem Grapher-Window der Regelbasis (Abbildung 6.5.) erscheint der Rule-System Editor.

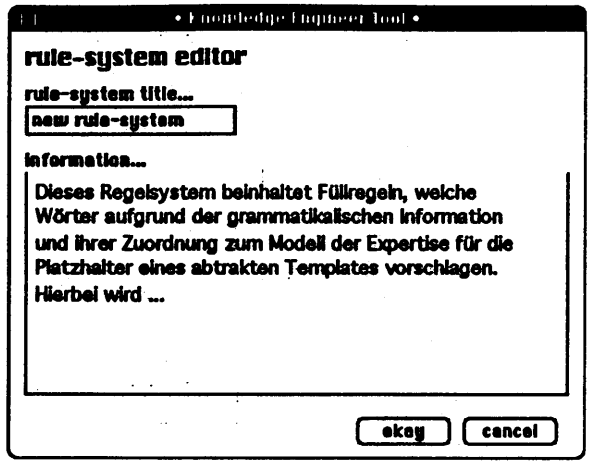


Abbildung 6.6.: Der Rule-System Editor

Durch Drücken des "Okay"-Button wird das neue Regelsystem in die Regelbasis eingefügt.

Jedem Regelsystem ist ein eigenes Grapher-Window zugeordnet, indem die Füllregeln (Bewertungs- und Filtrationsregeln) dargestellt werden. Diese Darstellung ist abhängig von dem aktuell ausgewählten View (semantic-view, syntactic-view,...).

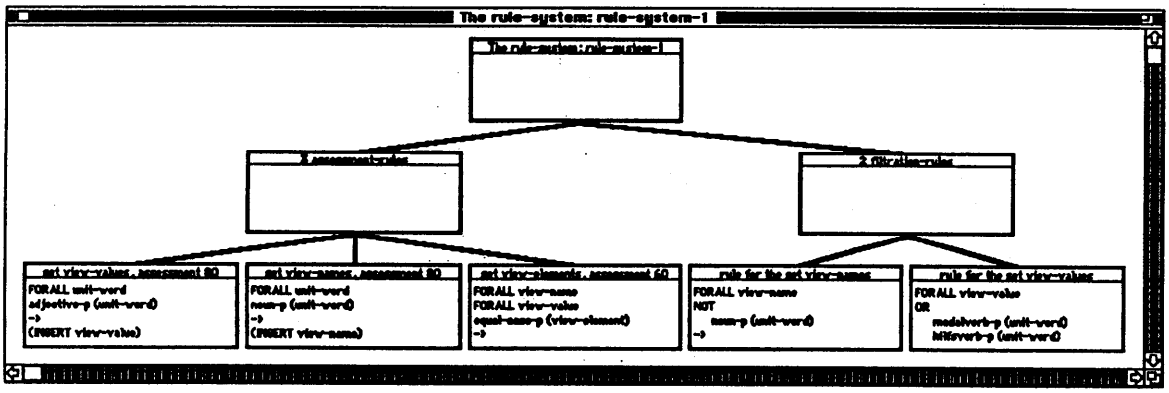


Abbildung 6.7.: Grapher-Window eines Regelsystems mit den hier enthaltenen Füllregeln

Für das Editieren der Füllregeln steht in INFOCOM ein Regeleditor zur Verfügung, in dem die Prämisse einer Regel durch das Verknüpfen von Prädikaten durch logische Operatoren festgelegt[9] wird.

[9] Zur Erinnerung: Bewertungsregel fügen Elemente in eine Menge ein, während Filtrationsregeln Elemente aus einer Menge entfernen. Damit steht die Konklusion einer Füllregel fest und ist nur von ihrem Typ abhängig.

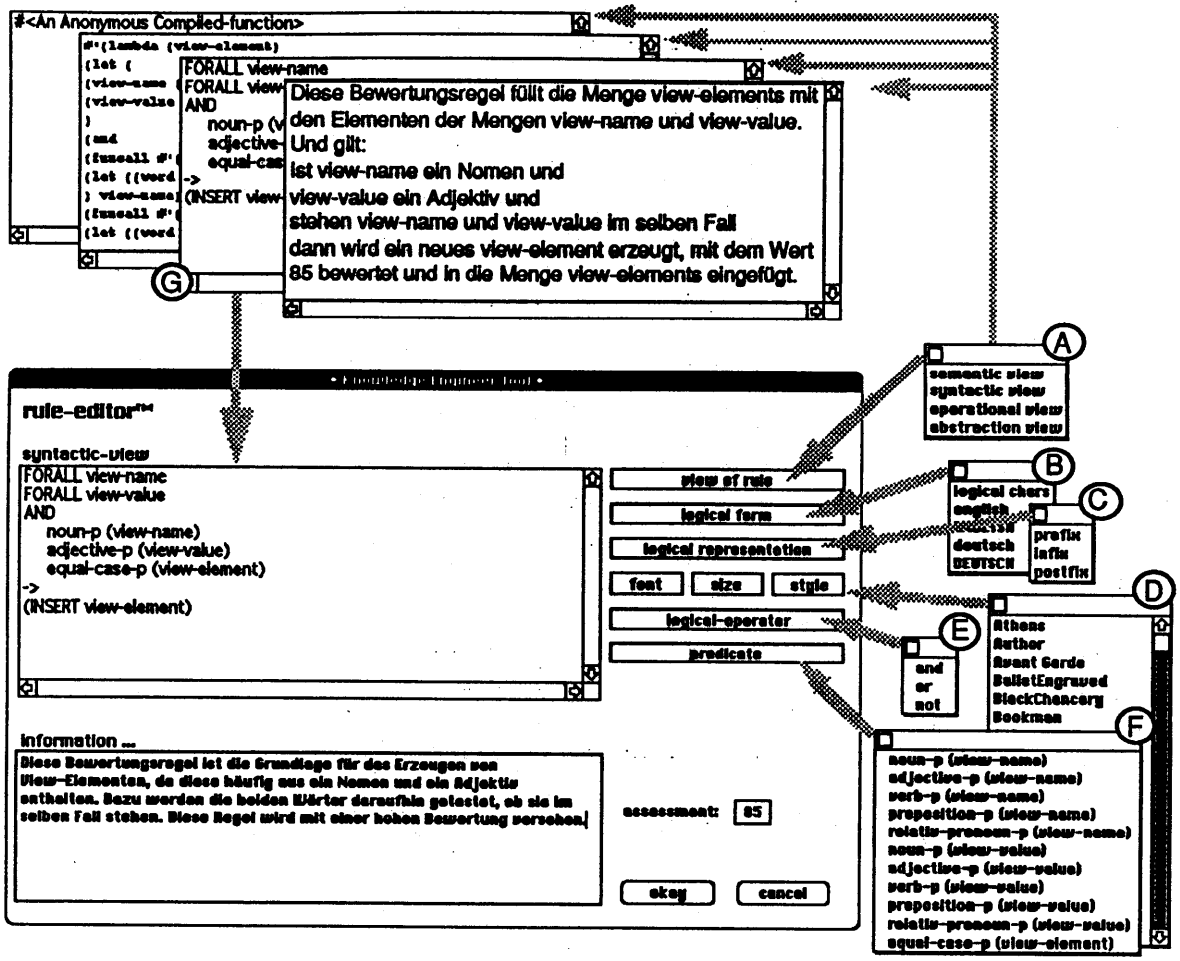


Abbildung 6.8.:Der Rule-Editor mit den vier Views
und den "Pull-Down" Menüs.

Das Editieren der Prämisse erfolgt im "syntactic-view" (A) durch selektieren der Buttons "logical-operator" und "predicate" in einer von drei Repräsentationsformen (C), z.B. in post-fix Notation. Hierfür werden in den Pull-Down-Menüs (E) und (F) die logischen Operatoren und *die* Prädikate aufgelistet, welche für die Elementtypen der Argumente der Füllregel zur Verfügung stehen.

Durch das Auswahlmenü (A) kann ein anderer View auf die zu editierende Füllregel ausgewählt werden, so daß diese beispielsweise im "operational-view" durch den Knowledge-Engineer verifiziert werden kann (G). In einem weiteren Eingabefeld kann dieser zusätzliche Informationen über die Füllregel formulieren.

Für eine Bewertungsregel kann außerdem ein Wert zwischen 0 und 100 eingegeben werden, der die Ausdruckskraft der Regel wiederspiegelt. Mit diesem Wert werden während des Füllalgorithmus auch die Elemente bewertet, welche die Prämisse der Bewertungsregel erfüllen.

In INFOCOM sind für die verschiedenen Elementtypen bereits mehrere Prädikate vordefiniert. Für die Eingabe neuer Prädikate steht ebenfalls ein Editor zu Verfügung. Die Arbeit mit diesem Prädikat-Editor soll im folgenden beschrieben werden.

6.2.3. Eingabe neuer Prädikate

Prädikate werden, wie in Kapitiel 5.1.2. beschrieben, über Elementtypen definiert. In dem Prädikat-Editor wird ein Prädikat durch eine Lisp-Funktion beschrieben.

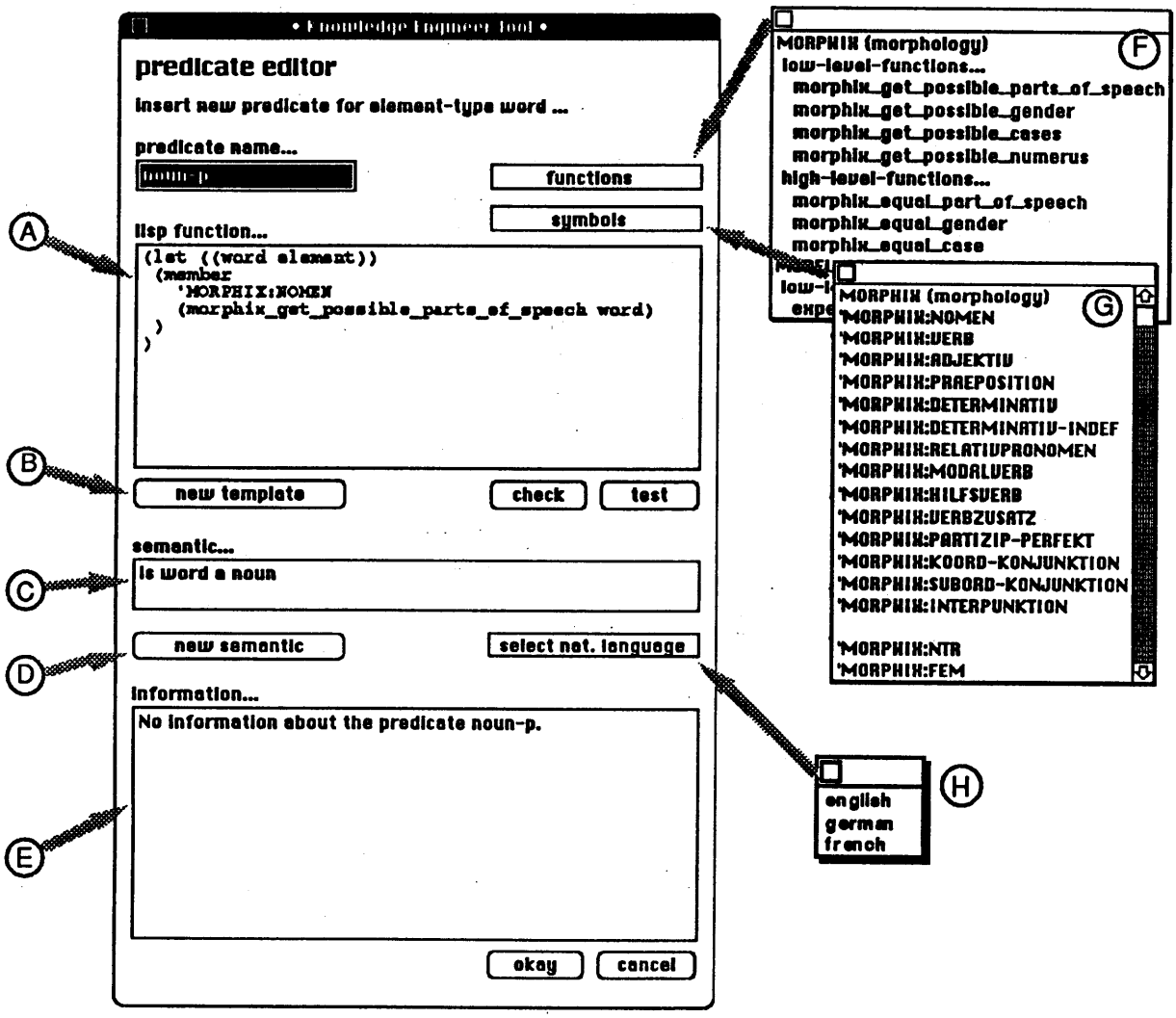


Abbildung 6.9: Der Prädikateditor mit den "Pull-Down"-Menüs

Die Eingabe eines neuen Prädikats ist in drei Schritte unterteilt:

1.) **Eingabe des Funktionsrumpfes**

Nachdem für das neue Prädikat ein Name eingegeben worden ist, wird mit (B) automatisch ein Funktionsrumpf generiert, welcher Zugriffsfunktionen auf die Sub-

Elementtypen des mit dem Prädikat assoziierten Elementtyps enthält. In Kapitel 5.1.2. ist dies am Beispiel des Elementtyps `view-element` dargestellt.
Die Menüs (G) und (F) zeigen Symbole und Lisp-Funktionen, welche neben den Standart-Lisp-Funktionen zur Verfügung stehen. Sie können durch das Selektieren mit der Maus an die aktuelle Position des Eingabefensters (A) kopiert werden.

2.) **Eingabe einer Semantik in natürlicher Sprache**

Der "semantic-view" zeigt die Bedeutung einer Füllregel in natürlicher Sprache. Um diesen View automatisch generieren zu können, wird jedes Prädikat durch eine Liste von Wörtern beschrieben. Für eine, durch (H) ausgewählte, natürliche Sprache werden in einem Dialog nacheinander Wörter erfragt, die vor, zwischen und nach den Argumenten des Prädikates aufgelistet werden sollen:
Beispiel:

Für den Elementtyp `view-element` soll ein Prädikat eingegeben werden, das testet, ob die Subelementtypen `view-name` und `view-value` im selben Fall stehen:

Prädikat:	`casus-equal-p(view-element)`
Semantik	"...stehen `view-name` und `view-value` im selben Fall..."

Da für jeden der beiden Füllregeltypen ("Diese Bewertungsregel...") und jeden logischen operator ("und", "oder") eine Beschreibung in einer natürlichen Sprache zur Verfügung steht, kann der "semantic-view" auf triviale Weise automatisch durch das Aneinanderhängen der Satzfragmente erzeugt werden.

3.) **Überprüfen und Testen der Funktion**

Nachdem die Syntax der Lisp-Funktion durch Drücken des Buttons "check" überprüft worden ist, kann diese getestet werden ("test"). Hierbei kann der Benutzer Elemente

7. Diskussion und Ausblicke

In diesem Kapitel soll eine Bewertung der Vorgehensweise von INFOCOM hinsichtlich verschiedener Kriterien versucht werden.

7.1. Beurteilung der Vorgehensweise

Ziel der in diesem Dokument vorgestellten Vorgehensweise ist die Unterstützung des Experten bei der Formalisierung informalen Wissens. Hierbei wird nicht der Anspruch erhoben, stets hundertprozentige Ergebnisse liefern zu können, wie es in Systemen zur automatischen Analyse natürlicher Sprache angestrebt wird. Vielmehr soll dem Experten durch interaktiven Dialog der Umgang mit dem für die Formalisierung notwendigen

7.2. Ergebnisse des Füllalgorithmus

Im folgenden sollen die Ergebnisse diskutiert werden, die der Füllalgorithmus für drei verschiedene Wissenseinheiten geliefert hat. Dazu werden für jede Wisseseinheit jeweils das vom Experten ausgewählte abstrakte Template, das durch ihn (manuell) gefüllte Template und das durch den Füllalgorithmus automatisch gefüllte Template angegeben. Die Zahlen, die bei den automatisch gefüllten Templates hinter den View-Elementen stehen, geben deren Platzierung an (in Abhängigkeit ihrer Bewertung berechnet). Die Abkürzung "ne" besagt, daß das entsprechende View-Element (bzw. Teile davon) manuell in dem Line-Editor nacheditiert werden mußte(n).

Für die erste Wissenseinheit hat der Füllalgorithmus auf Anhieb das gewünschte Ergebnis geliefert. Dieses gute Ergebnis ist damit zu begründen, daß zum einen der Text dieser Wissenseinheit kurz ist, d.h. nur wenige Wörter für eine geringe Anzahl zu füllender Passagen vorhanden sind, und zum anderen, daß die aufzufindenen Wörter "Rundrehen" und "Oberfläche" mit den Views general plan bzw. abstract product als Schlüsselwörter in dem Modell der Expertise assoziiert waren.

Wissenseinheit 1

"Beim Runddrehen entstehen überdrehte Oberflächen."

abstraktes Template

```
consequence(operator (M5WZ1)
            plan(SPLAN)
            -->
            product(APROD))
```

durch Experten gefülltes Template

```
consequence(operator (M5WZ1)
            Runddrehen()
            -->
            Oberfläche(überdreht))
```

Ergebnis des Füllalgorithmus

```
consequence(operator (M5WZ1)
            Runddrehen()                    (1)
            -->
            Oberfläche(überdreht))          (1)
```

In der zweiten Wissenseinheit waren die gesuchten Wörter "Schnittbedingungen", "Werkzeugstabilität" und "Eckenwinkel" ebenfalls mit den Views plan, environment und concrete plan als Schlüsselwörter assoziiert. Hier mußte der Experte lediglich das durch den

Algorithmus vorgeschlagene View-Element `Werkzeugstabilität(groß)` zu `Werkzeug(stabil)` umformen.

Wissenseinheit 2

"Wegen der bei extremen Schnittbedingungen erwünschten Werkzeugstabilität sollte der Eckenwinkel moeglichst groß sein."

abstraktes Template

```
precondition(operator (M5WZ1)
             plan(PLAN)
             environment(ENV)
             -->
             plan(CPLAN))
```

durch Experten gefülltes Template

```
precondition(operator (M5WZ1)
             Schnittbedingung(extrem)
             Werkzeug(stabil)
             -->
             Eckenwinkel(groß))
```

Ergebnis des Füllalgorithmus

```
precondition(operator (M5WZ1)
             Schnittbedingung(extrem)      (1)
             Werkzeug(stabil)              (1, ne)
             -->
             Eckenwinkel(groß))            (3)
```

Für den View concrete plan wurde das View-Element `Eckenwinkel(extrem)` auf den ersten Platz gesetz. Auch wenn einem Experten der Domäne klar sein mag, daß in diesem Zusammenhang mit einem *extremen* Eckenwinkel ein *großer* gemeint ist, muß bei der Formalisierung ein konkreterer Wert eingesetzt werden.

Bei der dritten Wissenseinheit konnte kein befriedigendes Ergebnis erzielt werden, obwohl diese nur aus einem relativ kurzen Satz besteht. Grund hierfür ist zum einen, daß "Freiwinkel" nicht in der Liste der Schlüsselwörter des Expertisenviews general plan vorkam. Zum anderen wird die zu dem "Freiwinkel" gehörende Gradangabe "4°" von keiner Füllregel als solche erkannt und bewertet, so daß es zu einer schlechten Platzierung des gesammten View-Elements kommt.

Wissenseinheit 3

"Harte Werkstoffe sind am besten mit geringen Freiwinkeln 4° zu bearbeiten."

abstraktes Template

```
precondition(operator (M5WZ1)
             product(APROD)
             -->
             plan(SPLAN))
```

durch Experten gefülltes Template

```
precondition(operator (M5WZ1)
             Werkstoff(hart)
             -->
             Freiwinkel(4°))
```

Ergebnis des Füllalgorithmus

```
precondition(operator (M5WZ1)
             Werkstoff(hart)              (1)
             -->
             Freiwinkel(4°))              (7)
```

Das oben genannte Problem tritt beispielsweise auch bei DIN-Bezeichnungen und Formeln auf, welche im Allgemeinen als gute Kandidaten für zu füllenden Passagen angesehen werden können. Für die durch die Testergebnisse aufgezeigten Schwachstellen des Füllalgorithmus sollen im folgenden Kapitel Verbesserungsvorschläge für diesen angegeben werden.

7.3. Ausblicke

Zur Verbesserung der von dem Füllalgorithmus gelieferten Ergebnisse müssen zusätzliche Füllregeln durch den Knowledge Engineer definiert werden. Eine Ontologie, die in COKAM+ integriert wird, soll das hierfür benötigte Hintergrundwissen liefern.

In INFOCOM werden die Wissenseinheiten durch einen *Preprozessor* für den Füllvorgang vorbereitet. Bisher werden hierbei beispielsweise Formel wie "E = M * C^2" in "E" "=" "M" "*" "C^2"" zerlegt. Daher sollten Mechanismen eingeführt werden, welche in dieser Phase Formeln detektieren, damit diese anschließend wieder zusammengestzt und später als solche erkannt werden können (testen durch Prädikate).

Ein weiters Problem ist durch die große Anzahl von Begriffen gleicher Bedeutung gegeben. Eine Normierung solcher Wörter (z.B. "extrem" für "auffällig", "ausgefallen", "ausgeprägt", "außergewöhnlich", usw.) führt zu einer exakteren Zielrepräsentation.

Im weiteren soll der Füllalgorithmus von INFOCOM mit umfangreicheren Regelsystemen getestet werden, wobei mit Hilfe der oben genannten Erweiterungen bessere Ergebnisse zu erwarten sind.

Literaturverzeichnis

[AlexanderFreiling+87] Alexander, J.H., Freiling, M.J., Shulman, S.J., Rehfuss, S., and Messick, S.L. Ontological Analysis: an ongoing experiment. *International Journal of Man-Machine Studies* 26(1987), 473 - 485.

[FinklerNeumann86] Finkler, W., Neumann, G., MORPHIX - Ein hochportabler Lemmatisierungsmodul für das Deutsche. Memo 8, Fachbereich Informatik, Universität Saarbrücken, FRG, 1986

[Huf94] Huf, H. Modellbasierte und fallorientierte Strukturierung für die schrittweise Formalisation von Wissen - Eine Untersuchung mit COKAM+. Projektarbeit am DFKI Kaiserslautern (1994, in press)

[LiddyJörgenson+92] Liddy, E.D., Jörgenson, C.L., Sibert, E., Yu, E. Sublanguage Grammar in Natural Language Processing for an Expert System.

[McDermott88] McDermott, steps towards a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic, Marcus, S., Ch. 8, pp. 225 -256, Boston, 1988.

Glossar

Expertise

Unter Expertise eines Menschen soll das spezielle Problemlösewissen und die Problemlösefähigkeit verstanden werden, die dieser Mensch zur Problemlösung innerhalb eines bestimmten Anwendungsgebietes erworben hat.

Fall

Ein Fall besteht aus der Beschreibung eines in der Domäne vorkommenden Problems bzw. einer Aufgabe und der Beschreibung einer Lösung zu dieser Aufgabe, die bereits erfolgreich angewandt wurde.

View

Als View werden die Klassen bezeichnet, die eine Ein- Ausgabeklasse in der domänenspezifischen Inferenzstruktur darstellen.

View-Element

Ein View-Element ist ein mit einem View des Modells der Expertise assoziiertes Wortpaar, welches aus einem view-name und einem view-value besteht.
Ein solches Wortpaar beschreibt den Zusammenhang zwischen einer Begrifflichkeit (view-name) der Zieldomäne und einem ihm zugeordneten Wert (view-value).

Wissenseinheit

Eine Wissenseinheit ist ein elementarer Teil des Domänenwissens.
Wissenseinheiten können informal, semiformal und formal repräsentiert werden.

Wissensingenieur (Knowledge Engineer)

Ein Wissensingenieur ist ein Experte auf dem Gebiet des Knowledge Engineerings. Er entwickelt direkt wissensbasierte Systeme und/oder ist daran indirekt durch die Entwicklung von Werkzeugen beteiligt.

Zielrepräsentation

Die Zielrepräsentation ist die formale Darstellung, in der das Domänenwissen beschrieben sein soll, wenn die Wissensakquisition durch den Experten beendet ist.

Anhang A: Ein Regelsystem

Set "view-names"

Diese Füllregel iteriert auf der Menge
unit-words.
Und gilt:
ist unit-word ein Nomen
dann wird aus unit-word
ein neues Element gebildet, mit 80 bewertet
und in die Menge view-names einfügen.

```
FORALL unit-words
   noun-p (unit-word)
   -->
   INSERT-ELEMENT (unit-word)
```

Diese Filtrationsregel iteriert auf der Menge
view-names.
Und gilt:
ist view-name kein Nomen
dann wird view-name aus der Menge
view-names entfernt.

```
FORALL view-names
   NOT
     noun-p (view-name)
   -->
   DELETE-ELEMENT (view-name)
```

Set "view-values"

Diese Füllregel iteriert über der Menge
unit-words.
Und gilt:
ist unit-word ein Adjektiv
dann wird aus unit-word
ein neues Element gebildet, mit 80 bewertet
und in die Menge view-values einfügen.

```
FORALL unit-words
   noun-p (unit-word)
   -->
   INSERT-ELEMENT (view-value)
```

Diese Filtrationsregel iteriert auf der Menge
view-values.
Und gilt:
ist view-value Nomen
oder
ist view-value ein Modalverb
oder
ist view-value ein Hilfsverb

dann wird view-name aus der Menge
view-names entfernt.

```
FORALL view-values
   OR
     noun-p (view-value)
     modal-verb-p(view-value)
     hilfs-verb-p (view-value)
   -->
   DELETE-ELEMENT (view-value)
```

Set "view-elements"

Diese Füllregel iteriert über den Mengen
view-names und view-values.
Und gilt:
ist view-name ein Nomen
und
ist view-value ein Adjektiv
dann wird aus view-name und view-value
ein neues Element gebildet, mit 80 bewertet
und in die Menge view-elements einfügen.

```
FORALL view-names
FORALL view-values
   noun-p (view-name)
   advective-p (view-value)
   -->
   INSERT-ELEMENT (view-element)
```

Diese Füllregel iteriert über den Mengen
view-names und view-values.
Und gilt:
stehen view-name und view-value im selben Fall
dann wird aus view-name und view-value
ein neues Element gebildet, mit 60 bewertet
und in die Menge view-elements einfügen.

```
FORALL view-names
FORALL view-values
   equal-case-p (view-name view-value)
   -->
   INSERT-ELEMENT (view-element)
```

Für die Mengen plan, product, environment, general plan, ..., concrete environment wurden Füllregeln definiert, welche die View-Elemente mit einer hohen Bewertung versehen, deren view-namen als Schlüsselwörter mit dem entsprechenden View im Modell der Expertise assoziiert sind. Für die Menge abstract product hat die Füllregel folgende Gestalt:

Set "abstract product"

Diese Füllregel iteriert über der Menge
view-elements.
Und gilt:
ist der view-name von dem view-element ein mit dem View abstract product assoziiertes Schlüsselwort

dann wird aus view-element
ein neues Element gebildet, mit 80 bewertet
und in die Menge abstract-product eingefügt.

```
FORALL view-elements
   keyword-p ('APROD view-element)
   -->
   INSERT-ELEMENT (abstract-product )
```

Anhang B: Ein Interface zu MORPHIX

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Interface : MORPHIX
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun get-morphix-functionality ()
  (let ((morphix (make-instance 'auxiliar-functionality
                                :name "MORPHIX (morphology)")))
    (insert-auxiliar-functionality morphix)

    (wt-functions
     (list " functions..."
           "  (morphix_get_possible_parts_of_speech word)"
           "  (morphix_get_possible_gender     word)"
           "  (morphix_get_possible_case       word)"
           "  (morphix_get_possible_nummerus   word)"
           " predicates..."
           "  (morphix_test_symbol_p           word  symbol)"
           "  (morphix_test_equal_symbol_p     word1 word2 symbol)"

           "  (morphix_test_part_of_speech_p   word  symbol)"
           "  (morphix_test_gender_p           word  symbol)"
           "  (morphix_test_case_p             word  symbol)"
           "  (morphix_test_nummerus_p         word  symbol)"

           "  (morphix_equal_part_of_speech_p word1 word2)"
           "  (morphix_equal_case_p           word1 word2)") morphix)
    (wt-symbols
     (list "'MORPHIX-NOMEN"
           "'MORPHIX:ADJEKTIV"
           "'MORPHIX:PRAEPOSITION"
           "'MORPHIX:DETERMINATIV"
           "'MORPHIX:DETERMINATIV-INDEF"
           "'MORPHIX:RELATIVPRONOMEN"
           "'MORPHIX:MODALVERB"
           "'MORPHIX:HILFSVERB"
           "'MORPHIX:VERBZUSATZ"
           "'MORPHIX:PARTIZIP-PERFEKT"
           "'MORPHIX:KOORD-KONJUNKTION"
           "'MORPHIX:SUBORD-KONJUNKTION"
           "'MORPHIX:INTERPUNKTION"
           " "
           "'MORPHIX:NTR"
           "'MORPHIX:FEM"
           "'MORPHIX:MAS"
           " "
           "'MORPHIX:NOM"
           "'MORPHIX:GEN"
           "'MORPHIX:DAT"
           "'MORPHIX:AKK"
           " "
           "'MORPHIX:SG"
           "'MORPHIX:PL") morphix)))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; MORHIX: Hilfsfunktionen
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setf *morphix_gender*  (list 'MORPHIX:NTR 'MORPHIX:FEM 'MORPHIX:MAS))

(defun tree-member (liste element)
  (cond ((null liste) nil)
        ((atom liste) (equal liste element))
        ((member element liste))
        (t
         (append (tree-member (car liste) element)
                 (tree-member (cdr liste) element)))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; MORHIX: Funktionen
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(defun morphix_get_possible_parts_of_speech (word)
  (cdr (tree-member (morphix::m word) 'MORPHIX:WORTART)))

(defun morphix_get_possible_gender   (word)
  (let ((tree (morphix::m word))
        (possible-gender nil))
    (dolist (gender *morphix_gender* possible-gender)
      (if (tree-member tree gender)
        (setf possible-gender
              (append possible-gender (list gender)))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; MORHIX: Prädikate
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun morphix_test_symbol_p (word symbol)
  (tree-member (morphix::m word) symbol))

(defun morphix_test_equal_symbol (word1 word2 symbol)
  (and
   (tree-member (morphix::m word1) symbol)
   (tree-member (morphix::m word1) symbol)))


(defun morphix_test_part_of_speech_p (word po-speech)
  (tree-member (morphix::m word) po-speech))

(defun morphix_test_gender_p (word gender)
  (tree-member (morphix::m word) gender))

(defun morphix_test_case_p (word case)
  (tree-member (morphix::m word) case))

(defun morphix_test_numerus_p (word numerus)
  (tree-member (morphix::m word) numerus))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(get-morphix-functionality)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
(defun morphix_equal_case_p (word1 word2)
  (let ((list-of-equal-case (morphix_get_equal_cases word1 word2)))
    (not (= 6 (count nil list-of-equal-case)))))

(defun morphix_get_equal_cases (word1 word2)
  (let ((cases1 (morphix_get_possible_cases word1))
        (cases2 (morphix_get_possible_cases word2)))
    (mapcar #'(lambda (x y)
                (intersection x y))
            cases1 cases2)))

(defun morphix_get_possible_cases (word)
  (let* ((tree    (morphix::m word))

         (mas     (tree-member tree 'MORPHIX:MAS))
         (fem     (tree-member tree 'MORPHIX:FEM))
         (ntr     (tree-member tree 'MORPHIX:NTR))

          (mas-sg (car (delete 'MORPHIX:SG (tree-member mas 'MORPHIX:SG))))
           (mas-pl (car (delete 'MORPHIX:PL(tree-member mas 'MORPHIX:PL))))
           (fem-sg (car (delete 'MORPHIX:SG(tree-member fem 'MORPHIX:SG))))
           (fem-pl (car (delete 'MORPHIX:PL(tree-member fem 'MORPHIX:PL))))
           (ntr-sg (car (delete 'MORPHIX:SG(tree-member ntr 'MORPHIX:SG))))
          (ntr-pl (car (delete 'MORPHIX:PL(tree-member ntr 'MORPHIX:PL)))))
    (list mas-sg mas-pl fem-sg fem-pl ntr-sg ntr-pl)))


(get-morphix-functionality)
```

Anhang C: Das Klassenkonzept

Das in dieser Arbeit vorgestellte System wurde in Lisp geschrieben. In diesem Kapitel sollen die CLOS[10]-Klassen und die auf ihnen definierten Methoden kurz beschrieben werde.

Klasse *Rule-Base*

Slots	Beschreibung
title	Der Titel der Regelbasis.
template-types	Die Templatetypen, die in COKAM+ als Vorgaben zur Formalisierung dienen.
element-types	Verschiedene, vordefinierte Elementtypen.
rule-systems	Die in der Regelbasis zur Verfügung stehenden Regelsysteme.
rule-base-window	Ein Grapher-Window, in dem die Regelbasis mit den Elementtypen und den Regelsystemen dargestellt wird.
rule-system-window-1 *rule-system-window-2*	Zwei Grapher-Windows, in denen die Füllregeln eines Regelsystems dargestellt und untereinander kopiert werden können.
element-type-window	Darstellung eines Elementtyps und den auf dem Elementtyp definierten Mengen und Prädikaten.
element-type-hierarchy-window	Ein Grapher-Window, das einen Elementtyp mit der Hierarchie sämtlicher Subelement-Typen zeigt.
set-hierarchy-window	Darstellung einer Menge mit allen Quellmengen als gerichteter Graph.
information	Informationen über die Regelbasis.

[10] CLOS (Common Lisp Object System) ist ein Teil der ANSI-Spezifikation von Common Lisp.

Dienste	Beschreibung
load-rule-base	Laden einer Regelbasis aus dem Ordner "rule-base".
save-rule-base	Abspeichern der aktuellen Regelbasis.
delete-rule-base	Löschen der aktuellen Regelbasis.
create-rule-base	Kreieren einer neuen Regelbasis.
show-rule-base	Anzeigen der aktuellen Regelbasis in einem Grapher-Window (rule-base-window).
edit-new-rule-system	Aufruf der Rule-System-Editors. Anschliessend wird das neue Regelsystem in die aktuelle Regelbasis eingefügt.
restore-rule-base	Wiederherstellen aller Objektreferenzen der Elemente der Regelbasis.

Klasse *Template-Type*

Slots	Beschreibung
name	Der Name des Templatetyps.
source-sets	Die mit dem Templatetyp assoziierten Grundmengen (z.B. "unit-words", "premise-view-symbols", "conclusion-view-symbols").
destination-sets	Die mit dem Templatetyp assoziierten Zielmengen (z.B. "premise-view-elements", "conclusion-view-elements").
finish-action	Eine Aktion, die nach dem Füllen eines abstrakten Templates ausgeführt werden soll.
print-function	Eine Funktion, welche die Darstellung des abstrakten Templates eines Typs bestimmt.
information	Informationen über den Templatetyp.
Dienste	**Beschreibung**
suggest-filled-templates	Aufruf des Füllalgorithmus.
pprint-template	Darstellung eines Templates aufgrund der print-function.

Klasse *Rule-Sytem*

Slots	Beschreibung
title	Der Titel des Regelsystems.
assessment-rules	Die Bewertungsregeln des Regelsystems.
filtration-rules	Die Filtrationsregeln des Regelsystems.
information	Durch den KE eingegebene Informationen über das Regelsystem

Dienste	Beschreibung
edit-rule-system	Editieren eines Regelsystems in dem Rule-System-Editor.
show-rule-system	Darstellen der Bewertungs- und Filtrationsregeln eines Regelsystems in einem Grapher-

Klasse *Rule*

Slots	Beschreibung
set	Die Menge, auf der die Füllregel definiert ist.
status	Der Status (aktiv/deaktiv) der Füllregel.
condition	Die Prämisse der Füllregel in einer internen Darstellung.
compiled-fn	Die compilierte Lisp-Funktion, die aus der internen Darstellung der Prämisse gewonnen wird. Während des Füllvorgangs wird diese Funktion mit den Elementen der Quellmengen als Argumente versorgt und ausgeführt.
information	Ein String, der zusätzliche (durch den KE eingegebene) Informationen über die Füllregel enthält.

Dienste	Beschreibung
edit-rule	Editieren einer Füllregel mit dem Regeleditor
generate-semantic-view	Liefert in einer Liste von Strings den "semantic-view" einer Füllregel in natürlicher Sprache

Unterklasse *Assessment-Rule*

Slots	Beschreibung
source-sets	Die Liste der Quellmengen
assessment	Die Bewertung der Regel.
action (instantiation class)	s.o. (InsertElement)

Klasse *Predicate*

Slots	Beschreibung
name	Der Prädikatname
element-type	Der Elementtyp, auf dem das Prädikat definiert ist.
function-body	Der in Lisp geschriebene Funktionsrumpf des Prädikates
function	Der aufgrund des Funktionsrumpfes generierte Lambda-Ausdruck
compiled-fn	Die compilierte Lisp-Funktion
semantics	Liste von Wörtern, die die Bedeutung des Prädikates in verschiedenen, natürlichen Sprachen beschreiben.
Dienste	**Beschreibung**
edit-predicate	Auruf des Prädikat-Editors.
compile-predicate	Kompiliert die Lisp-Funktion, die dem Prädikat zugeordnet ist.

Klasse *Set*

Slots	Beschreibung
name	Der Name der Menge.
elements	Die für die Menge durch den Füllalgorithmus erzeugten Elemente.
element-name	Der einem Element der Menge zugeordnete Name.
element-type	Der Elementtyp der Elemente einer Menge (eine Menge besitzt nur Elemente eines Typs).
source-sets	Die Quellmengen der Menge.

Klasse *Filling-Options*

Slots	Beschreibung
selected-rule-system	Das für den Füllalgorithmus aktuell selektierte Regelsystem.
number-of-suggested-templates	Die maximale Anzahl der gefüllten Templates, die in dem Auswahlfenster angezeigt werden sollen.
min-assessment	Schwellwert. Reduziert die Anzahl der angezeigten, gefüllten Template aufgrund ihrer Bewertung.
Dienste	**Beschreibung**
formalize-knowledge-unit	Aufruf des Hauptdialogs.
set-filling-options	Aufruf des Optionen-Dialogs.

Index

Deutsches
Forschungszentrum
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 gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or per 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.

DFKI Research Reports

RR-92-48
Bernhard Nebel, Jana Koehler:
Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi:
Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann:
Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo:
A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder:
How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist:
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi:
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit:
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz:
Satisfiability of the Smallest Binary Program
8 Seiten

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert:
Reasoning about Temporal Relations:
A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta:
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

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

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-93-44
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

DFKI Technical Memos

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

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

DFKI Documents

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

D-93-01
Philipp Hanschke, Thom Frühwirth: Terminological Reasoning with Constraint Handling Rules
13

[redacted]

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter: User Manual of COKAM+
23 pages

D-93-03
Stephan Busemann, Karin Harbusch(Eds.):
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster:
PPP: Personalized Plan-Based Presenter
70 pages

D-93-06
Jürgen Müller (Hrsg.):
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz, Saarbrücken, 29. - 30. April 1993
235 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch:
Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer:
TDL ExtraLight User's Guide
35 pages

D-93-10
Elizabeth Hinkelman, Markus Vonerden,Christoph Jung: Natural Language Software Registry (Second Edition)
174 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.):
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein:
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-14
Manfred Meyer (Ed.): Constraint Processing – Proceedings of the International Workshop at CSAM'93, July 20-21, 1993
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

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

Unterstützung des Experten bei der Formalisierung von Textwissen

INFOCOM - Eine interaktive Formalisierungskomponente

Frank Peters

D-93-26
Document