Erstellung einer deklarativen Wissensbasis über recyclingrelevante Materialien

Ulrich Buhrmann

April 1994

Die vorliegende Arbeit wurde vom Autor als Diplomarbeit an der Universität Kaiserslautern eingereicht.

Betreuung

Prof. Dr. Michael M. Richter

Dr. Harold Boley

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Daten und das Know-how von Unternehmen werden zunehmend in Informationssystemen gesammelt, gewartet und verfügbar gemacht. Da die in Datenbanken gespeicherten Daten das in einem Unternehmen vorhandene Wissen nur sehr unvollständig darstellen, sollten zukünftige Informationssysteme auch die Möglichkeit zur Darstellung von Wissen bieten. Dieses Wissen kann in Wissensbasen (Knowledge Bases, KBs) abgebildet werden. Solche Wissensbasen werden nicht nur für eine einzige spezielle Expertensystem-Anwendung erstellt, sondern für verschiedene Anwendungen genutzt, z. B. um ein Produkt zu konfigurieren, zu diagnostizieren, zu vermarkten und zu entsorgen. Daher muß das gespeicherte Wissen **deklarativ** formuliert sein, d. h., die Inhalte einer KB tragen schon als solche eine Bedeutung, die sich nicht erst aus der Verarbeitung ergibt. Weiterhin reicht die einmalige Erstellung solcher wiederverwendbarer KBs nicht aus. Das Wissen muß gewartet werden. Es ist im allgemeinen häufigen Verbesserungen und Ergänzungen unterworfen, die die Entwicklung des Unternehmens widerspiegeln.

1.1 Aufgabenstellung

Das Themengebiet der Validierung und Exploration von Wissensbasen durch globale Analyse (VEGA, Knowledge **V**alidation and **E**xploration by **G**lobal **A**nalysis) [BHH+92] wird an der Universität und am DFKI (**D**eutsches **F**orschungszentrum für **K**ünstliche **I**ntelligenz) in Kaiserslautern zur Verbesserung der oben beschriebenen Situation untersucht. So wird eine deklarative Darstellung entwickelt, mit der in hohem Maße wiederverwendbare KBs repräsentiert werden können. Weiterhin werden auf dieser Darstellung arbeitende Verfahren der Validierung (z. B. Konsistenz-/Vollständigkeitsprüfung) und der Exploration (z. B. Generalisierung von Mustern/Herstellung von Zusammenhängen) mittels globaler Analyse (z. B. abstrakte Interpretation/Datenflußanalyse) entwickelt und prototypisch zur Anwendung gebracht, mit denen die interaktive Wartung von KBs unterstützt werden soll. Exploration und Validierung werden dabei unter dem Begriff **Evolution** zusammengefaßt. Als Testobjekt für den deklarativen Formalismus (z. B. Regelverkettung) und

die Evolutions-Algorithmen (z. B. Faktengeneralisierung) sollen Teile einer bereits konzipierten KB zur **r**ecyclinggerechten **P**rodukt- und **P**roduktions**p**lanung (RPPP) [BBK93, Kre94] erstellt werden. Um die Anwendungsnähe der Verfahren zu demonstrieren, soll der gewählte RPPP-Bereich in einer späteren Ausbaustufe einen realistischen Umfang erhalten, so daß mit Hilfe der KB praxisnahe Probleme von echten Anwendungen gelöst werden können.

Aufgabe der vorliegenden Arbeit ist die prototypische Darstellung eines auszuwählenden Teilgebiets dieser KB, so daß erste Studien zum deklarativen Formalismus und der zu testenden Verfahren ermöglicht werden. Das zu formalisierende Teilgebiet sollte so reichhaltig sein, daß es einerseits isoliert, d. h. ohne weitere Teile der RPPP-KB, zu verwenden ist und andererseits erste Tests der Evolutions-Algorithmen erlaubt. In dem so ausgewählten Teilgebiet muß ein diese Anforderungen erfüllender Wissenskorpus erhoben werden, für den dann eine geeignete deklarative Darstellung zu finden ist. Evolutionsmöglichkeiten auf dieser KB sollen im fortlaufenden Text beschrieben werden.

Da sich die Repräsentationsformalismen gleichzeitig mit dem Aufbau der KB entwickeln, sollen Übersetzer in die jeweils aktuelle Darstellung implementiert werden, damit einmal formalisiertes Wissen stets automatisch in die Gegenwartsform gebracht werden kann.

1.2 Gliederung der Arbeit

In Kapitel 2 wird zunächst eine Übersicht des Konzepts der RPPP-KB gegeben, wobei erläutert wird, warum für den ersten Prototypen das Teilgebiet (recyclingrelevante) Materialien ausgewählt wurde. Weiterhin wird dort beschrieben, welche Quellen zur Wissenserhebung benutzt wurden. Die restlichen Abschnitte erklären, welche konkreten Anforderungen an die KB aus der Aufgabenstellung entstehen und welche verwandten Arbeiten berücksichtigt wurden.

Kapitel 3 beschreibt eine zuerst gewählte, ausdrucksmächtige objektzentrierte Darstellung des erhobenen Wissens. Im Anhang ist die erstellte KB über **r**ecyclierbare **T**hermo**plast**e (RTPLAST) in dieser Darstellung kommentiert abgedruckt.

In Kapitel 4 wird die Entwicklung einer sortierten hornlogischen Darstellung beschrieben, die deklarativer als die objektzentrierte ist, aber deren Vorteile (siehe Kapitel 3) für die KB trotzdem weitgehend beibehält. Die beiden Darstellungen werden verglichen, und ein Übersetzer von der objektzentrierten in die sortierte hornlogische Darstellung konzipiert, realisiert und anhand von RTPLAST demonstriert.

Das letzte Kapitel faßt die Ergebnisse dieser Arbeit zusammen und gibt eine Übersicht über die aus ihr resultierenden weiterführenden Fragestellungen.

In Anhang A ist die kommentierte objektzentrierte Version von RTPLAST abgedruckt. Anhang B enthält die aus der objektzentrierten Version automatisch erzeugte (daher unkommentierte) sortierte hornlogische Version von RTPLAST. Den Ausdruck eines Beispieldialogs, in dem beide Versionen verglichen werden, findet man

in Anhang C. Die Arbeit endet mit einem Glossar, in dem in dieser Arbeit und in RTPLAST verwendete Fachbegriffe erläutert werden.

Kapitel 2

Einordnung von RTPLAST

An die in dieser Diplomarbeit zu erstellende KB werden von verschiedenen Seiten unterschiedliche Anforderungen gestellt. Um diese Anforderungen erläutern zu können, muß die KB erst in ihre Entwicklungsumgebung eingeordnet werden. Deshalb beginnt dieses Kapitel mit einer Vorstellung der im Projekt VEGA konzipierten RPPP-KB, in deren Konzept RTPLAST ein Untermodul darstellt. Weiterhin wird in diesem Kapitel begründet, warum für die Erstellung des ersten Prototypen der Material-KB die Kunststoffe als Untergebiet ausgewählt wurden, und welche Erfahrungen bei der Implementierung bereits fertiggestellter Teilmodule gemacht wurden. In den weiteren Abschnitten wird erläutert, welchen Nutzen KI-orientierte Ansätze (KBs) im Vergleich zu Datenbanken bringen können, welche Anforderungen sich aus der deklarativen Wissensdarstellung der KB ergeben, und welche verwandten Arbeiten berücksichtigt wurden.

2.1 Die RPPP-Wissensbasis

In diesem Abschnitt wird eine Übersicht der Anwendungs-KB gegeben. Die einzelnen Module werden kurz erläutert, wobei das Modul Materialien genauer untersucht wird. Der an einer ausführlicheren Beschreibung interessierte Leser sei auf [BBK93, Kre94] verwiesen.

2.1.1 Einsatzbereiche einer Recycling-Wissensbasis

Nach Sondierung möglicher Bereiche für die KB, erscheinen uns Wissensteilgebiete zur **r**ecyclinggerechten **P**rodukt- und **P**roduktions**p**lanung (RPPP) [Bar91], im Rahmen eines Produktions- und Recycling-Planungs- und Steuerungssystems [CR91], als besonders geeignet. Nicht nur der hohe Anwendungsdruck (z. B. Elektronikschrott-Verordnung/Automobil-Recycling) bzw. der ökonomisch-ökologische Nutzen, sondern auch die reichhaltigen Strukturen und Zusammenhänge der Bewertung von Stoffkreisläufen bzw. Erstellung von Öko-Bilanzen haben zu dieser Themenwahl

geführt. Im Sinne von 'Knowledge Sharing/Reuse' [Bol90, NFF+91, Cze92, Kue93] sind verschiedene Verwendungsarten einer solchen RPPP-KB denkbar:

- Entscheidungsunterstützung für Produktingenieure bzgl. der Rohstoffauswahl, des Produktdesigns und des Produktionsprozesses [CR91].
- Inner- bzw. interbetrieblicher Vergleich der Recyclingmöglichkeiten eines bzw. mehrerer Unternehmen/Abteilungen.
- Die Umweltbeauftragten eines Unternehmens können durch Konsultation der Wissensbasis unter Verwendung des Evolutionssystems die Systematik ihrer Aktivitäten validieren und neue Möglichkeiten explorieren.
- Die Wissensbasis kann die Erstellung von kompletten Öko-Bilanzen für Produkte und Produktionsprozesse unterstützen.
- Rangfolgen für Produktdesigns und existierende Produkte, z. B. bzgl. 'Grad der Recyclierbarkeit' oder 'Energieverbrauch bei der Rückumwandlung' können erstellt werden.

Bei einer genügend breiten und deklarativen Bereitstellung des Wissens, sind weitere Einsatzbereiche möglich [SPM93, Ver93], auch solche, die nicht von vornherein vorgesehen waren.

2.1.2 Module einer RPPP-Wissensbasis

Die Fülle, der durch die intendierten Einsatzbereiche benötigten Informationen, setzt schon aus Komplexitätsgründen eine Modularisierung der RPPP-KB in Teilwissensbasen (Module) voraus (Abb. 2.1). Diese sollte sich möglichst auf natürliche Weise an bereits vorhandenen Datenbeständen orientieren.

Abb. 2.1 dient dazu, eine strukturierte Übersicht über wesentliche Module einer RPPP-KB und damit Referenzpunkte für unsere eigenen Beiträge zu geben.

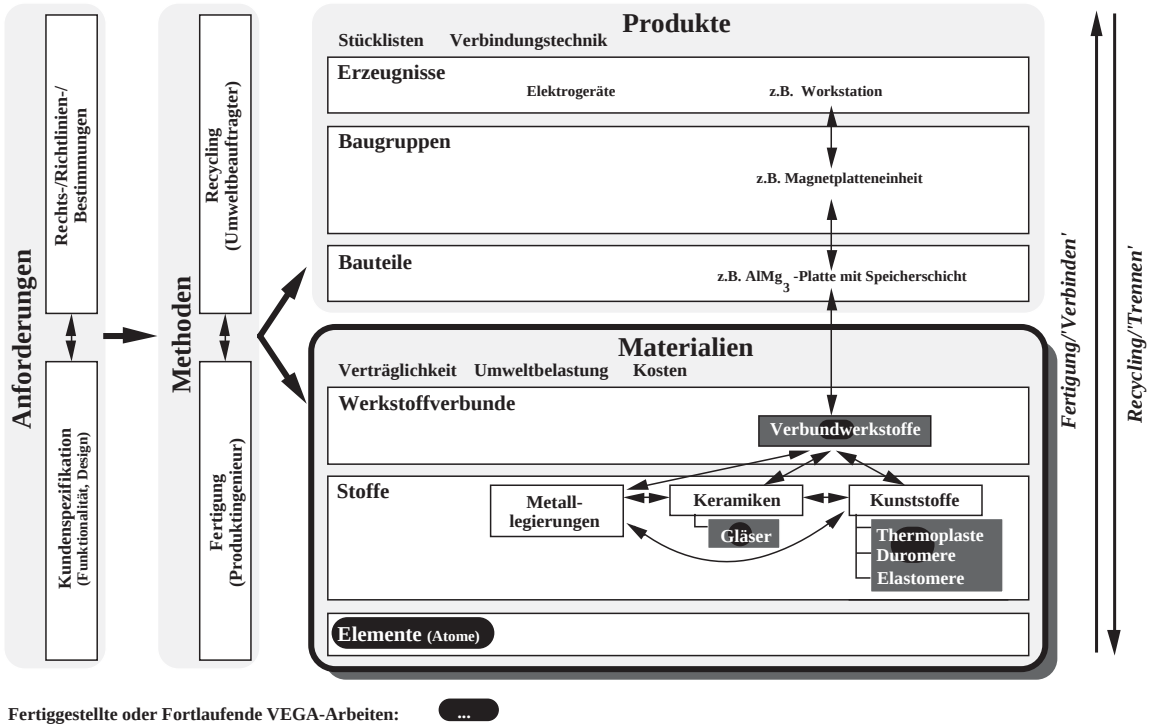

Abbildung 2.1: Die Material-KB im Kontext der RPPP-KB

In dem Modul *Anforderungen* werden einerseits die Kundenwünsche bzgl. der Funktionalität und des Designs des zu planenden Produkts, andererseits die durch Gesetze oder Richtlinien verordneten Auflagen, die das Produkt und der Produktionsprozeß erfüllen müssen, spezifiziert. Das Modul *Methoden* beinhaltet die der Unternehmung zur Verfügung stehenden Fertigungsmethoden (Unternehmenspotential). Sie werden im wesentlichen durch den Produktingenieur festgelegt. Parallel dazu kann der Umweltbeauftragte, durch Konsultation des Untermoduls *Recycling*, eine Recyclingstrategie festlegen. In dem Modul *Produkte* wird der Produktaufbau über die verwendeten *Baugruppen* und *Bauteile* definiert. Nach dem Produktgebrauch werden den Baugruppen oder Bauteilen, je nach möglichem Auflösungsgrad, die zuvor festgelegten Recyclingstrategien zugeordnet. Die zur Konstruktion der Produkte benötigten Materialien werden in dem Modul *Materialien* repräsentiert (siehe Abschnitt 2.1.3). Die Module sind derart konzipiert, daß schon während der Produktentwicklung recyclingrelevante Sachverhalte berücksichtigt werden können.

Beim Aufbau der RPPP-KB ist es sinnvoll, mit einem gut formalisierbaren und wiederverwendbaren Teilbereich zu beginnen. Wir haben uns für das Modul *Materialien* und das Untermodul *Stoffe* entschieden, da einerseits umfangreiche Datenbestände existieren und andererseits mehrere Module (*Werkstoffverbunde, Bauteile*) unmittelbar darauf zugreifen. Das Untermodul *Stoffe* stellt eine um recyclingrelevantes Wissen (z. B. Stoffverträglichkeiten, Umweltbelastung, Kosten) erweiterte Stoff-

Datenbank dar. Die Erweiterungen werden insbesondere durch KI-Repräsentationstechniken realisiert: Statt in einem fest formatierten, extensionalen Datenbanksystem, wird das Wissen in einem frei formatierten, intensionalen Informationssystem mit Inferenzregeln verfügbar gemacht; so sind z. B. Stoffsubstitutionsvorschläge für Querverweise auf und Vergleiche zwischen Stoffen repräsentierbar.

Aufbauend auf dem Untermodul *Stoffe* kann das Untermodul *Werkstoffverbunde* realisiert werden. Wissen über die sortenreine Verbind-/Trennbarkeit von Stoffen wird von Verträglichkeitsmatrizen, Diagrammen und natürlichsprachlichen Richtlinien auf eine einheitliche deklarative Repräsentation abgebildet.

2.1.3 Das Modul *Materialien*

Aufgabe des Recyclings ist die Wiedergewinnung von neuen Materialien aus alten, schon benutzten. Die Materialien stellen somit die 'Substanz' des Recyclings dar, wodurch es sinnvoll erscheint, eine modulare RPPP-KB mit dem Aufbau einer Material-KB zu beginnen. Außerdem sollte der Grundstein der RPPP-KB bereits reichhaltig genug sein, um ihn auch alleinstehend verwenden zu können. Dies gilt vor allem unter dem Gesichtspunkt, daß aus Kapazitätsgründen im Projekt VEGA nicht alle Module der RPPP-KB realisiert werden können. Auch um eine größtmögliche Nützlichkeit zu ermöglichen, wurden uns von Experten die Repräsentation von Materialien nahegelegt und z. B. die Abbildung spezieller Materialkreisläufe im Produktionsprozeß als weniger sinnvoll bezeichnet.

Durch die Vielfalt, der für das Recycling in Betracht kommenden Materialien, bedingt sich eine Unterteilung des Moduls *Materialien* in mehrere Untermodule. Wie Abb. 2.1 zeigt, besteht das Modul *Materialien* aus den Untermodulen *Werkstoffverbunde*, *Stoffe* und *Elemente*. Diese Aufteilung ergibt sich aus den unterschiedlichen Eigenschaften der einzelnen Teile der Module mit der Sichtweise eines Werkstoffkundlers, wobei nicht nur Recyclingeigenschaften betrachtet werden, so daß sich die KB dual auch für die im Projekt IMCOD[1] fokussierten Design- und Fertigungszwecke nutzen läßt (Wiederverwendbarkeit).

2.1.3.1 Das Untermodul *Werkstoffverbunde*

Das Modul *Werkstoffverbunde* befindet sich im Moment noch in der Startphase. In Angriff genommen wird es in einem gemeinsamen Projekt des **I**nstitut für **V**erbund**w**erkstoffe (IVW) in Kaiserslautern und des DFKI, in dem zum einen das anspruchsvolle Problem des Verbundwerkstoffrecyclings mit Hilfe von KI-Methoden zu untersuchen sein wird, das zum anderen aber auch die Materialauswahl des Produktingenieurs während des Produktentwurfs unterstützt.

Da in diesem Projekt dann die beiden bis dahin realisierten Untermodule des Materialmoduls (Abschnitte 2.1.3.2 und 2.1.3.3) verwendet werden, war beim Auf-

[1] Das Projekt IMCOD (Intelligent Manager for Comprehensive Design) [RBB+93] befaßt sich mit der Entwicklung eines *Designmanagres*.

bau von RTPLAST darauf zu achten, daß vor allem Wissen in die KB aufgenommen wird, daß für den Produktingenieur von Relevanz ist. Wegen der angestrebten Wiederverwendbarkeit sollte das Wissen allerdings so repräsentiert werden, daß es auch für andere Domänen (z. B. Kunststoffentwicklung) nutzbringend ist.

2.1.3.2 Das Untermodul *Stoffe*

Das Modul *Stoffe* gliedert sich wiederum in die drei Untermodule *Metallegierungen*, *Keramiken* und *Kunststoffe*. Im Projekt VEGA werden davon die Module *Kunststoffe* und *Keramiken* in Angriff genommen. Besonders wichtig erschienen uns dabei die Kunststoffe (Thermoplaste), da:

1. Genügend Wissen über Kunststoffe in Form von Literatur [KLW93], Experten (IVW) und bestehenden Datenbanken (CAMPUS) [BDS90] existiert.

2. Das Recycling von Kunststoffen noch nicht auf breiter Basis verstanden ist, somit unser Beitrag für das Recycling besonders wertvoll sein könnte.

3. Mit dem IVW ein Anwender vorhanden ist, so daß wir unter ständiger Benutzerkontrolle die KB anwendungsnah und nicht am Experten 'vorbei' modellieren können.

4. In Deutschland und speziell in der Region Rheinland-Pfalz ein starkes industrielles Interesse an Kunststoffen besteht.

Beim Aufbau der KB haben wir uns zunächst damit begnügt, die kleine, überschaubare Klasse von Kunststoffen darzustellen, die bereits in einer solchen Qualität recycelt werden, daß mit dem Recyclat neue, gleichwertige[2] Produkte konstruiert werden können.[3] Da es im Augenblick hauptsächlich nur Verfahren zum Recyceln von Thermoplasten gibt, entstammen die von uns gewählten Kunststoffe dieser Klasse. Abb. 3.1 zeigt unsere Expansion des Untermoduls *Thermoplaste*.

Grundlage für die Implementierung der KB war eine Teilmenge von COLAB [BHHM], einem am DFKI entwickelten Tool zur Wissensrepräsentation. Die Teilmenge sollte im Hinblick auf ihre Eignung zur Evolution ausgewählt werden (siehe Abschnitt 2.2). Da die Auswahl der Teilmenge zeitlich parallel zum Aufbau der KB erfolgte, also beide Prozesse aufeinander einwirkten, wurden automatische Übersetzer benötigt, um die KB stets auf dem neuesten Stand der Wissensrepräsentation zu halten. Die entsprechenden Übersetzer werden im Kapitel 3 erläutert.

Für die Wissenserhebung zu RTPLAST wurden verschiedene Quellen benutzt:

[2]Unter gleichwertig verstehen wir die Verwendung des Rezyklats in solcher Weise, daß aus einem ehemaligen Gehäuse auch wieder ein solches wird (eigentliches Recycling) und der Kunststoff beispielsweise nicht als Füllstoff in der Bauindustrie verwendet wird (Downcycling). Die Rezyklate dürfen allerdings zur Qualitätssicherung einen Gewissen Anteil neuer Kunststoffe enthalten.

[3]Für eine Liste derartiger Kunststoffe sei der Leser auf [KLW93] verwiesen.

- Es fanden informelle Gespräche mit Experten (vor allem Mitarbeiter des IVW) statt, aus denen wir verschiedene Schlüsse zur Wissensrepräsentation zogen. Für die Darstellung von ingenieurspezifischem Wissen werden in der KB hauptsächlich mechanische, thermische und verarbeitungsspezifische Eigenschaften der Kunststoffe abgebildet, die um einige Recyclingeigenschaften erweitert wurden.

- In der Fülle der Literatur zu diesem Thema konnten wir uns auf ein Buch konzentrieren [KLW93], welches die 'Grundlage' der KB bildet. Die Wahl fiel auf dieses Buch, weil:

  1. es ein sehr aktuelles (Erscheinungsjahr 1993) und reichhaltiges Standardwerk zum Thema "Konstruieren in Kunststoffen" ist;
  2. in vielen Kapiteln das duale Thema Recycling berücksichtigt wird.

  Dem Buch von Kunz, Land und Wiener haben wir neben allgemeinen Informationen über die Verarbeitung von Kunststoffen die Bedeutung ihrer verschiedenen Eigenschaften für die Konstruktion entnommen. Weiterhin beruht die Auswahl der von uns repräsentierten Kunststoffe auf einer Liste, die Teil des Buches ist und Kunststoffe enthält, die bereits in einer garantierten Qualität als Recyclat verkauft werden. Schließlich haben wir eine dort abgebildete Kunststoffhierarchie zum Teil auf unsere KB übertragen.

- In der CAMPUS-Datenbank [BDS90] werden von fast allen deutschen Herstellern Daten zu den von ihnen hergestellten Kunststoffen abgelegt. Diese Datenbank wurde von uns analysiert. Wir kamen zu dem Schluß, daß die CAMPUS-Datensätze auf Fakten abgebildet werden können. Beim Ausbau der KB wäre es denkbar, die Daten automatisch in eine Teilmenge unserer Repräsentation zu übersetzen. Durch die KI-technischen Repräsentationen (Regeln, Sortenhierarchien etc.) und Verarbeitungsmöglichkeiten stellt unsere KB somit eine echte Obermenge der CAMPUS-Datenbank dar.

- Herstellerprospekten und -datenblättern haben wir schließlich zusätzliche Kunststoff-Informationen entnommen. So wurden in CAMPUS nicht enthaltene Fakten, Regeln und Sortenhierarchien hinzugefügt und Freitexte formalisiert.

2.1.3.3 Das Untermodul *Elemente*

Das Modul *Elemente* ist eine KB über das vollständige Periodensystem der chemischen Elemente [SSB91]. Mit dem Aufbau der KB wurde schon in dem früheren DFKI-Projekt ARC-TEC begonnen und mit Beginn des VEGA-Projekts fortgefahren.

Implementiert wurde die KB direkt in RELFUN [BEH+93], einem Teilsystem von COLAB. RELFUN ist eine funktional-logische Programmiersprache, deren Klauseln

in Hornlogik abgebildet werden können; sie ist somit auch nach PROLOG, in diesem Fall sogar nach purem PROLOG, übersetzbar. Eine LISP- oder PROLOG-artige Syntax kann vom Benutzer frei gewählt werden, wodurch die Portierung der KB in beide Sprachwelten erleichtert wird.

Beim Aufbau der Elemente-KB stellte sich eine Synthese von Rapid-Prototyping und CLASSIC [BMPS+90] als geeignetste Entwicklungsmethode, in bezug auf unsere Problemstellung (Darstellung von Stoffwissen in RELFUN), heraus. Während der Konzipierung von RTPLAST wurden von uns zwar noch verschiedene andere Designmethoden, z. B. KADS [vHB92], auf ihre Anwendbarkeit untersucht, doch ergab die Analyse, daß – wegen der positiven Erfahrungen – die Synthese von Rapid-Prototyping und CLASSIC den anderen Methoden vorzuziehen war. Bei dieser Synthese wird in einem ersten CLASSIC-Zykel ein Prototyp erstellt, der dann in weiteren Zykeln ausgebaut wird [wg93]. Neben dem Vorteil, daß beim Rapid-Prototyping getroffene Entscheidungen schnell überprüft werden können, war für uns von besonderer Bedeutung, daß der von uns erstellte Elemente-Prototyp mit geringem Aufwand modifiziert werden konnte, was bei der sich im Wandel befindlichen Repräsentationssprache und der Offenheit bzgl. multipler Anwendungen eines unserer Ziele darstellte. Ein Problem war hierbei jedoch die normale 'positionale' Darstellung des Faktenwissens in der Elemente-KB. Änderungen oder Erweiterungen der Darstellung der Fakten machten entsprechende Änderungen in den auf die Fakten zugreifenden Regeln notwendig. Für dieses Problem logik-orientierter Repräsentationen mußte eine flexiblere, objektzentrierte Darstellung gefunden werden, die in Kapitel 3 beschrieben wird.

2.2 Anforderungen an RTPLAST

Der Aufbau einer KB wird in der Regel kein isolierter Prozeß sein; so wird meistens zu der KB eine Inferenzkomponente entwickelt, die zusammen mit der KB ein Expertensystem bildet. RTPLAST ist sogar in einem noch größerem Kontext zu sehen. Es soll nicht nur eine konkrete Anwendung erstellt werden (Materialauswahl), vielmehr ist das Modul *Kunststoffe* Teil einer größeren KB-Konzeption (RPPP-KB) und soll im Themengebiet VEGA als Testbett für die Explorations-/Validierungs-Methoden und die VEGA-Repräsentationssprache benutzt werden. Weiterhin müssen beim Aufbau äußere Bedingungen beachtet werden. Unter äußeren Bedingungen sind hierbei die jeweils vorhandenen Wissensrepräsentationstools und die Rechnerausstattung sowie die sich ändernden Nutzeranforderungen zusammengefaßt.

2.2.1 Die Anwendung in der Verbundwerkstoff-Domäne

Das erste Einsatzgebiet von RTPLAST wird die Unterstützung der Materialauswahl eines Verbundwerkstoffkonstrukteurs sein (siehe Abschnitt 2.1.3.2). Die konkrete Problemstellung sieht folgendermaßen aus: Bei der Verarbeitung eines mit Glasmatten verstärkten Thermoplasten (GMT) wird der Prepreg (GMT-Halbzeug) zuerst in ei-

nem Ofen erhitzt. Anschließend wird der nun formbare Prebreg aus dem Ofen herausgenommen und in eine Presse gelegt, in der er seine endgültige Form erhält. Problematisch wird dieser Vorgang, wenn sehr große Formteile zu pressen sind und mit dem der Presse zur Verfügung stehendem Druck ein Prebreg nicht in alle Teile der Form gedrückt werden kann, bevor er wieder abkühlt. Da dem Konstrukteur in der Regel nur eine einzige Presse zur Verfügung steht, er den Druck also nicht uneingeschränkt verändern kann, bleiben ihm nur zwei Lösungen für dieses Problem:

1. Er erhöht die Anzahl der Prebregs und verteilt diese gleichmäßig in der Form. Die Folge ist, daß zum Verteilen der Prebregs mehr Zeit benötigt wird. Da Zeit im Produktionsbetrieb ein sehr kostbares Gut ist, kann diese Lösung nur selten (z. B. für Einzelanfertigungen) angewendet werden.

2. Da Kunststoffe unterschiedliche Zähigkeiten haben, kann der Ingenieur versuchen einen GMT zu finden, der weniger zäh ist, sich also in der Presse besser und schneller verteilen läßt. Das Problem hierbei ist, daß bei dieser Substitution diejenigen Eigenschaften des GMT erhalten bleiben sollen, die der Konstrukteur bei der ursprünglichen Materialauswahl berücksichtigt hatte. Da aber viele Eigenschaften sich gegenseitig beeinflussen, z. B. ist bei geringerer Zähigkeit auch die Temperaturbeständigkeit kleiner, ist dies kein triviales Problem.

Ansätze zur Lösung 2 wollen wir mit unserer KB unterstützen. Es sollte dabei allerdings nicht außer acht gelassen werden, daß ein grundlegender Anspruch an RTPLAST ihre Wiederverwendbarkeit ist. Sie ist nicht nur Teil der RPPP-Konzeption, sondern soll auch eigenständig weiterverwendet werden; so ist die KB z. B. die Grundlage einer geplanten Erstellung eines Expertensystems zur recyclinggerechten Konstruktion von Computergehäusen, das im Rahmen einer Diplomarbeit in Zusammenarbeit des DFKI und IBM implementiert werden soll.

Welche Inhalte RTPLAST zur Lösung des GMT-Problems haben sollte, wurde bereits in Abschnitt 2.1.3.2 beschrieben. Für eine mögliche Wiederverwendung sind folgende Punkte zu beachten:

- Das in der KB enthaltene Wissen sollte so dargestellt werden, daß es für spätere Anwender leicht abrufbar ist. Es muß also ohne detaillierte Kenntnis der internen Struktur der KB abfragbar sein.

- Durch eine deklarative Darstellung (siehe Abschnitt 2.2.2) kann das 'Verstecken' von Wissen in Kontrollstrukturen vermieden werden.

- Die KB sollte leicht erweiterbar sein. Eine Erweiterung darf also ebenfalls keine zu detaillierte Kenntnis der KB voraussetzen; dies gilt vor allem dann, wenn die KB eine bestimmte Größe erreicht hat. Weiterhin sollten keine zu großen Änderungen an bestehenden Teilen nötig sein.

2.2.2 Deklarativität

Die Aufgabe dieser Diplomarbeit ist die Erstellung einer deklarativen KB über recyclingrelevante Materialien. Die Vorteile der deklarativen Darstellung sollen dem Leser an dieser Stelle kurz erläutert werden. Die Übersetzung von 'deklarativ' lautet 'beschreibend'. Für eine deklarative Wissensrepräsentation bedeutet dies, daß mit ihr Wissen unabhängig von der Art der Verarbeitung dargestellt wird.

Ein Beispiel für eine deklarative Darstellung ist Hornlogik, da es keine Rolle spielt, in welcher Reihenfolge die Klauseln benutzt werden. Man kann natürlich auch eine in Hornlogik abgebildete KB in einer bestimmten Reihenfolge abarbeiten, wie es z. B. in Prolog gemacht wird. In Prolog kann es passieren, daß, durch die Tiefensuche, ein Programm nicht logisch vollständig ist, oder direkt durch die `Cut`- und `Once`-Operatoren Lösungen abgeschnitten werden. Beschränkt man sich allerdings auf pures Prolog/DATALOG, d. h. den Hornlogik-Kern, so kann in Prolog durchaus eine deklarative KB erstellt werden. Nicht deklarativ sind z. B. in imperativen Programmiersprachen erstellte 'KBs', da hier die Abarbeitungsreihenfolge einen ganz massiven Einfluß auf die Lösungen für Anfragen hat.

Warum sollte RTPLAST deklarativ sein?

Datenverarbeitende Systeme oder Wissensstrukturen ohne spezifizierte deklarative Semantik neigen dazu, Teile des in ihnen enthaltenen Wissens in Kontroll- und Inferenz-Algorithmen zu 'verstecken'. Die semantische Analyse kann also nicht statisch sondern nur mit anfragenbasierten Input-/Output-Operationen erfolgen. Die Evolution von Wissen ist mit solchen Operationen aber nur sehr unbefriedigend durchzuführen. Eine nicht deklarative KB wäre also als Testbett der Evolutions-Algorithmen denkbar ungeeignet.

Weiterhin wird durch eine klar definierte Semantik der Ausgangs-KB sichergestellt, daß durch interaktive Exploration gefundene Wissenseinheiten in den richtigen Zusammenhang zu den bestehenden Wissenseinheiten gebracht werden können.

Ebenso sollte zu einer solchen Evolutions-KB eine Meta-KB existieren, in der Wissen über die Repräsentation der Evolutions-KB enthalten ist, damit die Evolutionsergebnisse vor dem richtigen Hintergrund interpretiert werden können. Bei einer nicht deklarativen KB müßten hier also die Kontroll- und Inferenz-Algorithmen abgebildet werden, wofür keine befriedigende Lösung existiert.

Ein weiterer Vorteil der deklarativen Darstellung ist die erleichterte Wiederverwendbarkeit. Es ist klar, daß es einem späteren Benutzer leichter fallen wird, eine KB zu verstehen und zu verwenden, wenn in ihr kein Wissen 'versteckt' wird, sondern explizit beschrieben ist. Ähnliches gilt für neue Software- oder Hardware-Plattformen.

Ein die oben genannten Ansprüche erfüllender Sprachkern für die VEGA-Repräsentationssprache wäre DATALOG, d. h. pures Prolog ohne Konstruktoren. Ein weiterer Vorteil von DATALOG ist, daß bereits eine größere Anzahl von Algorithmen zur Exploration auf DATALOG-KBs existieren, die zur Evolution eingesetzt werden könnten. Da schließlich DATALOG als Grundlage der deduktiven Datenbanken angesehen werden kann, werden deduktive Datenbanken und die in sie übertragbaren

relationalen Datenbanken den Evolutions-Operationen zugänglich. Dies hat nicht nur den Vorteil, daß dann auch relationale Datenbanken analysiert werden können, sondern umgekehrt auch Verfahren zur Analyse von relationalen Datenbanken einsetzbar sind [KMK91].

Die Verwendung von DATALOG steht allerdings im Widerspruch zu in Abschnitt 2.2.1 beschriebenen Anforderungen. So sind für Anfragen an eine DATALOG-KB Kenntnisse über die interne Struktur nötig; d. h. der Anwender muß wissen in welcher Relation und an welcher Stelle der Relation er den geforderten Attributwert findet. Erweiterungen stellen ebenfalls ein Problem dar, da zusätzliche Attribute in einer Relation Änderungen im Datenbankschema nötig machen. Für RTPLAST wurde dieses Problem dadurch gelöst, daß die KB in einer von uns erstellten objektzentrierten Repräsentationssprache implementiert wurde, die automatisch in eine DATALOG-ähnliche Sprache übersetzt wird. Dies wird in Kapitel 3 erläutert.

Im Themengebiet VEGA sind folgende Erweiterungen von DATALOG geplant:

1. Konstruktoren
2. endliche Domänen
3. Sortenhierarchien[4]

Wie diese Erweiterungen genau aussehen, wird in Kapitel 4 beschrieben. Das Kapitel enthält außerdem Konzepte zur Überführung der objektzentrierten Darstellung in die Erweiterungen.

2.2.3 Allgemeine Anforderungen

Mit 'Allgemeinen Anforderungen' ist die Einhaltung von in der Informatik üblichen Prinzipien gemeint. Darunter fallen so grundlegende Dinge wie die genaue Dokumentation der KB und die softwaretechnische Aufteilung in verschiedene Module ab einer bestimmten Größe.

Da RTPLAST zu einem Großteil auf Daten basiert, die direkt aus einer Datenbank entnommen sind, sollen auch aus diesem Gebiet Prinzipien übernommen werden. Gerade über die Validierung von Datenbanken ist schon viel nachgedacht worden [Eve86, Ull88], so daß einige der in Datenbanken üblichen Verfahren auch in VEGA berücksichtigt werden können. Hier sind vor allem Kardinalitäts- und Integritätsprüfungen gemeint.

Für die Erstellung einer großen Datenbasis ist es weiterhin sinnvoll, einen effizienten Zugriff auf die Daten und eine zentrale (d. h. möglichst redundanzfreie) Datenhaltung sicherzustellen.

[4] Mit den Sortenhierarchien können z. B. KL-ONE-artige KBs den Evolutions-Methoden zugänglich gemacht werden.

2.3 Verwandte Arbeiten

Zu dem Thema Material-KBs und Künstliche Intelligenz existieren bereits verschiedene Arbeiten. Ein Teil dieser Arbeiten [LG89, HFRF90, HPFR87, vdVM91, PB93, PSK+93] wurde von uns analysiert und Ergebnisse wurden in unserer Arbeit an entsprechenden Stellen übernommen. Die wichtigsten Arbeiten werden in diesem Abschnitt kurz vorgestellt.

Die erste Arbeit beschreibt das *Cyc*-Projekt [LG89] von Lenat und Guha bei der Microelectronics and Computer Technology Corporation (MCC). Lenat und Guha sind der Auffassung, daß zur Abbildung von intelligenten menschlichen Verhalten im Rechner, wie z. B. Humor, Phantasie usw., keine "Patentlösung" existiert; d. h. es existiert kein Lernalgorithmus, mit dem aus wenigen Wissenseinheiten viel neues Wissen abgeleitet werden kann. Vielmehr meinen sie, daß diese Eigenschaften des Menschen aus der Fülle seines Wissens resultieren. Um Aspekte des Menschen im Rechner nachahmen zu können, d. h. den Rechner ein wenig intelligenter zu machen, wollen sie eine KB erstellen, die eine genügend große Menge Wissen enthält, um ihre Theorie zu beweisen.

Für uns ist diese Arbeit von Interesse, da:

1. Ein Teil des in *Cyc* abgebildeten Wissens Materialien beschreibt.

2. Die *Cyc*-KB sehr groß werden soll. Daher haben sich Lenat und Guha ausführlich mit der Darstellung von großen Wissensmengen befaßt. Hinzu kommt, daß sie die Verwendung des Wissens nicht genau planen können, weshalb sie eine in hohem Maße wiederverwendbare KB erstellen müssen.

3. Die objektzentrierte *CycL*-Sprache eignet sich gut zur Abbildung von (Material-) Ontologien.

Ihre Vorgehensweise in diesen drei Punkten sollte also auf unser Problem übertragbar sein.

In [HFRF90, HPFR87] wird das Projekt ALADIN von Hulthage et al. an der Carnegie-Mellon University beschrieben. ALADIN ist ein Expertensystem, das Metallkundler beim Entwurf neuer Aluminiumlegierungen unterstützt. Metallurgische Daten und Konzepte werden in der Form von Schemata in einer deklarativen Weise repräsentiert. Diese Schemata werden mit `Is-a-`, `Instance-`, `Subset-of-` und `Member-of-`Relationen in eine Hierarchie gegliedert. Nach dem Studium der oben genannten Papiere erschien uns die Übertragung der Schemata – die wir abgewandelt, d. h. objektzentriert, verwendet haben – und ihrer Gliederung auf unsere Kunststoff-Domäne als sinnvoll.

Dieses Projekt erschien uns wichtig, weil hier in Zusammenarbeit von Informatikern und Domänen-Experten ein Material-Expertensystem geschaffen wurde, das seine Anwendbarkeit in der Praxis nachgewiesen hat.

Beide KBs wurden anwendungsorientiert entwickelt, ein Anspruch der auch an unsere KB gestellt wird. Beide Projekte sind für uns deshalb besonders lehrreich. An

dieser Stelle soll festgehalten werden, wie die RTPLAST-Repräsentation aussehen sollte, wenn man die in den beiden Projekten gemachten Erfahrungen berücksichtigt:

1. Sie sollte objektzentriert sein.
2. Die Objekte sollten in einer Hierarchie angeordnet werden.
3. In der Hierarchie sollte es einen Vererbungsmechanismus geben.

Diese Anforderungen lassen sich nicht ohne weiteres in DATALOG realisieren. Für RTPLAST wurde deshalb eine Repräsentationssprache entwickelt, die obige Punkte berücksichtigt und in der die KB implementiert wurde. Diese Version der KB wurde anschließend automatisch in eine positionale Form übersetzt (siehe Kapitel 3) und kann somit den Evolutions-Methoden zugänglich gemacht werden.

Kapitel 3

Die Repräsentation des Wissens

In diesem Kapitel wird die Repräsentation des für RTPLAST in der Kunststoffdomäne gesammelten Wissens beschrieben. Um die Repräsentation anhand von Beispielen erläutern zu können, enthält der erste Abschnitt dieses Kapitels die Beschreibung eines Ausschnitts des in RTPLAST enthaltenen Wissens. In diesem Abschnitt wird ebenfalls aufgezeigt, welche Wissensarten – wie Regeln, Fakten und Tabellen – RTPLAST enthält. In den folgenden Abschnitten wird die objektzentrierte Darstellung, die Abfrage/Verarbeitung und die Positionalisierung dieses Wissens beschrieben. Die letzten Abschnitte dieses Kapitels befassen sich mit dem als Teil der KB definierten Wissensbankschema und dem konkret in RTPLAST repräsentierten Wissen.

3.1 Der Ausschnitt aus der Kunststoffdomäne

Wie in Kapitel 2 beschrieben, enthält RTPLAST Wissen über Kunststoffe, wie es ein Ingenieur bei der Konstruktion von recyclinggerechten Produkten verwendet.

Prinzipiell wird in der KB jeder Kunststoff als Objekt aufgefaßt, das durch die Menge seiner Eigenschaften beschrieben werden kann. Diese Eigenschaften lassen sich durch Attribute mit zugeordneten Werten beschreiben. Die Zuordnung der Werte zu den Attributen erfolgt auf drei verschiedene Arten:

1. Sie werden dem Attribut direkt zugewiesen, z. B. hat jeder Kunststoff eine bestimmte Dichte.

2. Sie werden über eine Methode berechnet, z. B. kann ein Attributwert durch eine Interpolationstabelle beschrieben werden, so daß der konkret geforderte Wert erst noch interpoliert werden muß.

3. Sie werden über Regeln bestimmt; z. B. hängen die Verwendungsmöglichkeiten von der Art der Verarbeitung ab. Natürlichsprachlich sieht eine solche Regel wie folgt aus:

Wenn die Verarbeitungsweise von Kunststoff X gleich Spritzgießen ist, dann kann ein Computergehäuse erstellt werden.

Betrachtet man nun die Ansammlung dieser Objekte als KB, so wird sie der Forderung nach möglichst redundanzfreier Speicherung nicht gerecht, da es immer Gruppen von Kunststoffen gibt, die in vielen Attributen übereinstimmen und sich nur an einigen Stellen unterscheiden. Diese Kunststoffgruppen wurden deshalb zu einer Klasse zusammengefaßt, die ihre gemeinsamen Attributwerte von einer Oberklasse vererbt bekommen. Alternativ könnte man auch durch einen (generalisierenden) Evolutionsprozeß Attribute wie Dichte von der Klasse Kunststoffe zur obersten Klasse unserer Hierarchie, Materialien, hochverlagern und Computergehäuse durch das allgemeinere Elektrogerätegehäuse in der Beispielregel ersetzen.

Durch diese Art der Zusammenfassung gibt es zwei unterschiedliche Klassenarten in unserer Repräsentation. Zum einen sind das diejenigen Klassen, die konkret existierende Objekte der Domäne beschreiben. Diese Klassen werden *individual class* genannt. Zum anderen gibt es Klassen, in denen nur Attribut-Wert-Paare zusammengefaßt sind, die an tiefer in der Hierarchie liegende Klassen vererbt werden. Diese Klassen beschreiben also keine existierenden Objekte der Domäne, weshalb sie als *prototype class* bezeichnet werden. Da diese beiden Klassenarten eine unterschiedliche Semantik haben, sollten sie auch in der Wissensrepräsentation unterschieden werden. Hierdurch entsteht in unserer Repräsentation die für objektorientierte Systeme unübliche Unterscheidung zweier Klassenarten.

Für beide Klassenarten werden Instanzen definiert. Um auch diese unterscheiden zu können, werden die Instanzen der Individual-Klassen als *individual* und die der Protoyp-Klassen als *prototype* bezeichnet. Da für Prototyp-Klassen in RTPLAST jeweils nur ein Prototyp definiert ist, sind die Prototypen in Abb. 3.1 nicht dargestellt.

Abb. 3.1 ist ein Ausschnitt aus einer Grafik, die die komplette RTPLAST-Struktur beschreibt und im Anhang abgedruckt ist. Die in den folgenden Abschnitten zur Verdeutlichung verwendeten Beispiele beziehen sich auf Teile dieser Ausschnitts-KB. Die konkrete Strukturierung dieses Ausschnitts in Prototyp- und Individual-Klassen kann der Abbildung entnommen werden.

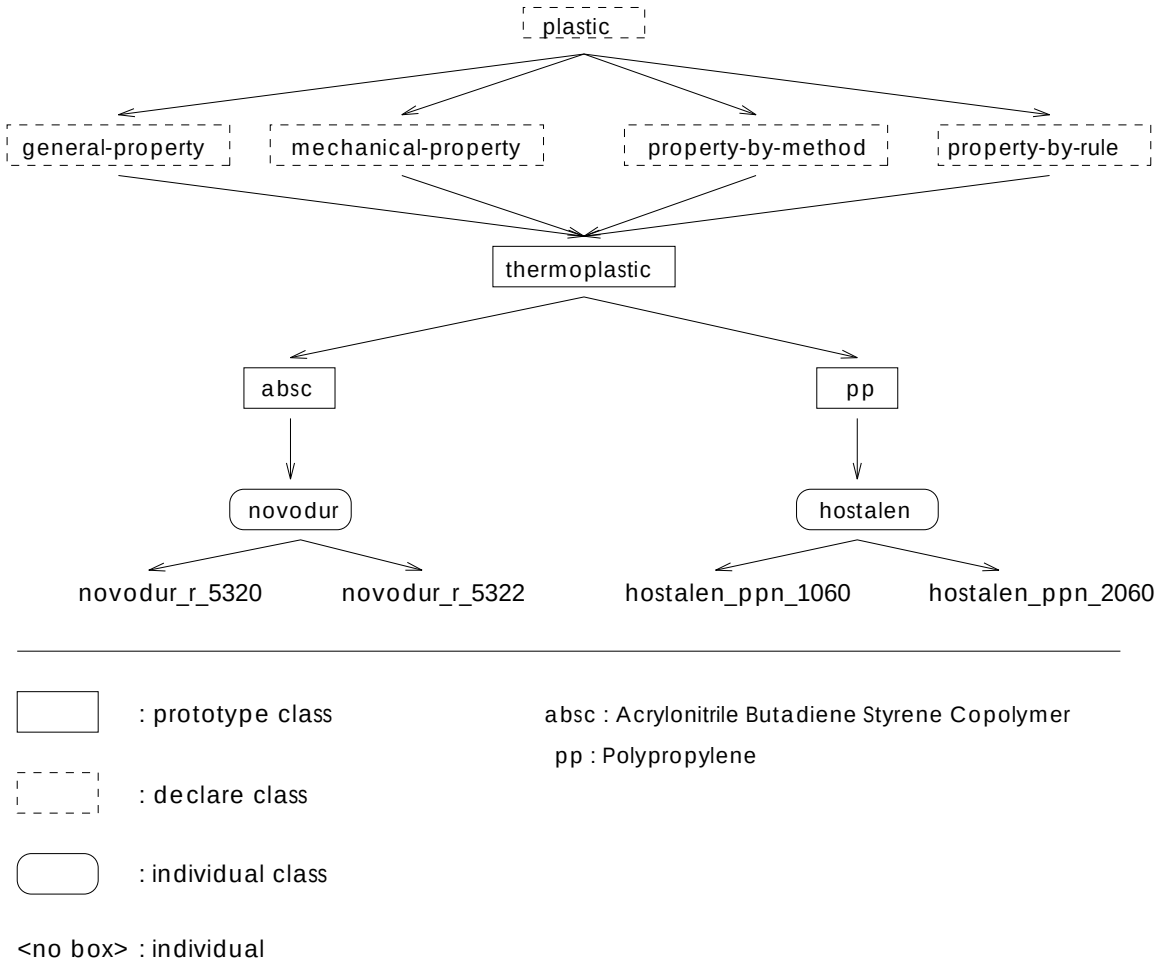

Abbildung 3.1: Vererbungshierarchie der Ausschnitts-KB

An dieser Stelle ergibt sich die Frage, ob es zwischen den Kunststoffen nicht auch Beziehungen gibt, die über Gemeinsamkeiten in den Attributwerten hinausgehen und die deshalb ebenfalls in die KB aufgenommen werden sollten. Solche Beziehungen sind z. B. in Frame-Systemen, wo ein Slot auf einen anderen verweisen kann, oder in Concept-Sprachen, wo Rollen zwischen den Konzepten definiert werden können, abbildbar. Die Analyse der Kunststoffdomäne ergab jedoch, daß solche Beziehungen bei der Materialauswahl nur eine geringe Bedeutung haben, weshalb sie in RTPLAST nicht abgebildet wurden. Da aber Beziehungen zwischen den einzelnen Attributen existieren, so nimmt z. B. die Zähigkeit der Kunststoffe mit zunehmender Temperaturbeständigkeit ebenfalls zu, sollten sie in einer späteren Ausbaustufe von RTPLAST integriert werden, um sie bei anderen Anwendungen oder zur Evolution nutzen zu können. Hierfür können die in Kapitel 4 eingeführten Sorten verwendet werden, mit denen die hornlogisch nachbildbaren Teile der Funktionalität von Concept-Sprachen (z. B. existenzquantifizierte Rollen) realisiert werden können.

Neben der Klassifikation nach gleichen Attributwerten ist die Aufgliederung der Attribute in Attributgruppen sinnvoll, da ein späterer Benutzer bestimmt nicht in einer sehr langen Liste von Attributen die von ihm gewünschten Attributwerte heraussuchen will. Die KB enthält deshalb sogenannte *declare class*, in denen gleichartige Attribute nur eingeführt und ohne Werte vererbt werden. Hierdurch gewinnt RTPLAST insgesamt an Übersichtlichkeit. Welche Arten von Attributen zu jeweils einer Klasse zusammengefaßt werden, kann Abb. 3.1 entnommen werden.

3.2 Die objektzentrierte Darstellung

Wie in Kapitel 2 beschrieben, sollte die Repräsentationssprache, in der RTPLAST implementiert wird, Objektzentrierung erlauben. Da als Kern der evolutionsgerechten Sprache aber DATALOG vorgesehen ist, mußte die objektzentrierte Sprache von uns erst konzipiert und prototypisch implementiert werden.

Das Ergebnis unserer Überlegungen war die Sprache ORF (Object-Centered RELFUN) [Sin93], die eine Erweiterung der Sprache RELFUN [BEH+93] ist. Da RELFUN eine Grundlage der VEGA-Sprache ist, wird sie im nächsten Abschnitt kurz beschrieben.

3.2.1 Kurzeinführung in RELFUN

RELFUN ist eine logisch-funktionale Programmiersprache, die u. a. als ein Teilsystem des hybriden Wissensrepräsentationstools COLAB [BHHM] verwendet wird, das in dem früheren DFKI-Projekt ARC-TEC entwickelt wurde. RELFUN kann mit einer LISP- oder Prolog-artigen Syntax benutzt werden. Da Hornlogik-Klauseln in den relationalen, Prolog-ähnlichen Teil von RELFUN abgebildet werden können, kann RELFUN als eine Grundlage der evolutionsgerechten Sprache dienen, denn, wie in Abschnitt 2.2.2 erläutert, soll DATALOG der Kern der VEGA-Sprache sein.

Ein durch die Verwendung von RELFUN/Prolog entstehendes Problem ist die dort benutzte Tiefensuche ('ODER-Sequentialität'), wodurch es zu Terminationsproblemen kommen kann, falls die KB nicht terminierende Pfade enthält. Da das Terminationsproblem auf das Halteproblem zurückgeführt werden kann, ist die automatische Entdeckung von nicht terminierenden Pfaden nicht möglich. Bei der Erstellung von RTPLAST wurde deshalb darauf geachtet, solche Pfade zu vermeiden. Ein anderes Problem ist die Nicht-Kommutativität der Literale ('UND-Sequentialität'), das aber in RTPLAST nicht zum Vorschein kommt.

Weiterhin stören die in RELFUN erlaubten Eingriffe in den Programmablauf. So sind die aus Prolog bekannten Konstrukte `once` und `cut` erlaubt, die nicht deklarativ sind. Bei der Implementierung von RTPLAST wurde deshalb auf den Einsatz dieser Konstrukte verzichtet.

In RELFUN ist die Definition von (nicht-deterministischen) Funktionen erlaubt, was bei einigen Anwendungen leichter verständliche Implementierungen ermöglicht,

als dies mit normalen Prologklauseln möglich wäre. Die Verwendung von Funktionen in RTPLAST ist aber problematisch, da die Evolutions-Algorithmen auf Relationen arbeiten. Funktionen können in RTPLAST trotzdem verwendet werden, da sie automatisch in Relationen umwandelbar sind. Bei der Relationalisierung wird aus jeder n-stelligen Funktion eine n+1-stellige Relation erzeugt, wobei die Rückgabewerte an der zusätzlichen (ersten) Stelle eingesetzt wird. Ein Beispiel hierfür sieht in der Prologsyntax wie folgt aus:

```
Funktion:
how-recyclable(Plast) :-
         contains(Plast, flamability-trigger) &
         only-in-closed-circle.

Relation:
how-recyclable(only-in-closed-circle, Plast) :-
         contains(Plast, flamability-trigger).
```

Der Rückgabewert der Funktion wird durch das vorangestellte Kaufmanns-UND gekennzeichnet und gibt in diesem Fall an, daß ein Flammhemmer enthaltender Kunststoff nur in geschlossenen Kreisläufen recycliert werden kann. Die Relation enthält nun eine zusätzliche Stelle, an der der ursprüngliche Rückgabewert eingetragen ist.

RELFUN kann im Interpreter- oder Emulatormodus benutzt werden. Für den Emulator muß die KB allerdings erst in eine compilierte Version übersetzt werden, die dann in einer **W**arren **A**bstract **M**achine (WAM) [War83, Nys85] abgearbeitet wird. Die Übersetzung der ORF- in die RELFUN-Darstellung ist eine Erweiterung der Compilation, was zur Folge hat, daß die ORF-KB nur im Emulatormodus benutzt werden kann. Es wäre jedoch möglich, auch den Interpreter entsprechend zu erweitern.

3.2.2 Die Repräsentationssprache ORF

Es folgt die Beschreibung der eigentlichen Repräsentation. Die Sprache ORF [Sin93] bietet hierfür verschiedene Konzepte und Methoden, die in den folgenden Abschnitten beschrieben werden. Sie hat Ähnlichkeiten mit den 'Feature'-Logiken, wie z. B. LogIn [AKN86] oder LIFE [AL88].

3.2.2.1 Die Attribut-Wert-Paare

Der Leser möge sich an dieser Stelle erinnern, daß eine Anforderung an die Sprache zur Implementierung von RTPLAST die leichte Erweiterbarkeit und Abfrage des enthaltenen Wissens war. Der Benutzer sollte also in der Lage sein, Anfragen an die KB zu stellen, ohne ihre komplette Struktur zu kennen. Wie in Abschnitt 2.2 erklärt,

wird eine positionalisierte Hornlogik-Relation diesem Anspruch nicht gerecht, wenn man bedenkt, wie viele Attribute einen Kunststoff beschreiben. Die Lesbarkeit der Relationen könnte verbessert werden, wenn Strukturen zur Gliederung der Relation benutzt würden.[1] Doch müßte der Benutzer diese Gliederung genau kennen, wenn er Attributwerte abfragen will.

Die Erweiterung der KB wäre in der positionalen Darstellung ebenfalls sehr aufwendig, da, falls ein Attribut zu einem Kunststoff hinzugefügt werden soll, alle Regeln der KB, die auf die Relation zugreifen, geändert werden müßten. Bei einer sehr großen KB kann dies sehr mühsam sein und jeder spätere Benutzer wird gründlich püfen, ob er diesen Aufwand auf sich nimmt, oder lieber eine neue KB anlegt, womit unsere KB also nicht mehr wiederverwendbar wäre.

Eine Lösung für dieses Problem ist die Darstellung der Attribute als zweistellige Relationen, wobei der Name des Attributs dem Namen der Relation entspricht, eine Stelle der Relation den Kunststoff identifiziert und die andere Stelle den Wert des Attributs für den identifizierten Kunststoff angibt. Konzepte zur Überführung der objektzentrierten in die sortierte zweistellige Darstellung findet der Leser in Kapitel 4. Dort werden auch gründlicher die Vor- und Nachteile der Darstellung diskutiert.

Wir haben uns bei der ersten Implementierung von RTPLAST für eine andere Lösung entschieden. Wobei darauf geachtet wurde, daß die bisher implementierten Teile so gestaltet sind, daß sie möglichst automatisch in andere sinnvolle[2] Repräsentationen übersetzt werden können.

3.2.2.2 Beschreibung von Individual-Klassen mit variabel langen Attribut-Wert-Paaren

In diesem Abschnitt wird die in ORF realisierte Beschreibung von Objekten durch Attribut-Wert-Paare erklärt. Hierfür wird erläutert, wie die zu beschreibenden konkreten Objekte der Domäne zu sogenannten Individual-Klassen zusammengefaßt, und die Klassen zusammen mit einer Menge von zulässigen Attributen deklariert werden.

Die Deklaration einer Klasse von Individuen soll hier an einem Beispiel aus der Ausschnitt-Domäne eingeführt werden:

```
declare(indi-class[novodur,
                   super[absc],
                   identifier, density,
                   tension_module_of_elasticity]).
```

[1] In diesem Fall wäre die KB nicht mehr DATALOG-gerecht, da Strukturen in DATALOG nicht zugelassen sind, in der Datenbankterminologie wäre die KB also nicht mehr in der ersten Normalform. Da in der VEGA-Sprache Strukturen als Erweiterung vorgesehen sind, könnten sie für RTPLAST jedoch verwendet werden.

[2] Mit sinnvoll ist hier die Erfüllung der in Abschnitt 2.2 spezifizierten Anforderungen gemeint.

Dies ist also eine mögliche Deklaration für die in Abbildung 3.1 eingeführte Klasse `novodur`. Novodur ist ein Produktname für von der Firma Hoechst hergestellte Thermoplaste. Wie dem Beispiel zu entnehmen ist, wird hier eine `indi-class` deklariert, wobei `indi` für `individual` steht; die Klasse ist also eine Zusammenfassung von konkreten Objekten der Domäne.

Durch den Eintrag `super[absc]` wird deklariert, daß die Klasse `novodur` eine Unterklasse der Klasse `absc` ist, wobei `absc` eine Abkürzung für eine bestimmte Kunststoffart ist, nämlich die Acrylonitrile-Butadiene-Styrole-Copolymere. Die Beziehung und die Bezeichner können zur Verdeutlichung in Abbildung 3.1 nachgeschaut werden. Der Begriff **Unterklasse** ist hierbei im folgenden Sinn zu interpretieren: Wie bereits beschrieben, sollte die RTPLAST-Repräsentation Möglichkeiten zur Vererbung von Attributwerten enthalten. Dies wurde so realisiert, daß Kunststoffe mit teilweise gleichen Eigenschaften zu einer Klasse zusammengefaßt werden. Für diese Klasse wird dann eine **Oberklasse** deklariert, zu der eine Definition existiert, in der die zu vererbenden – d. h. gleichen – Werte definiert werden. Wie diese Oberklassen deklariert und definiert werden, wird im folgenden Abschnitt genauer beschrieben.

In der letzten Zeile der Deklaration werden schließlich die für diese Klasse zulässigen Attribute eingeführt. Dies sind im einzelnen:

`identifier`: werden in jeder indi-class benutzt und kennzeichnen das konkrete Objekt.

`density`: ist die Dichte des Kunststoffs.

`tension_module_of_elasticity`: ist das sogenannte E-Modul. Es gibt die mechanische Belastung des Kunststoffs für einen normierten Prüfkörper an.

Das E-Modul und die Dichte wurden für dieses Beispiel ausgewählt, da dies vom Ingenieur häufig benutzte Attribute sind.

An dieser Stelle sollen nun zwei Kunststoffe definiert werden:

```
novodur(
  identifier[novodur_r_5320],
  density[0.84],
  tension_module_of_elasticity[2000]).

novodur(
  density[0.91],
  identifier[novodur_r_5322]).
```

Es ist leicht zu erkennen, daß die Attribute durch Kommata getrennt werden, und jedem Attribut in den folgenden eckigen Klammern ein Wert zugewiesen wird.

Der Leser kann an diesem Beispiel erkennen, wie Individuen der Klasse `novodur` im Prinzip wie Prolog-Relationen notiert werden. Da die eckigen Klammern in RELFUN zur Realisierung von passiven Konstrukten zugelassen sind, stellen die obigen Definitionen sogar syntaktisch korrekte RELFUN-Konstrukte dar. Durch die vorherige Deklaration von novodur als Klasse bekommen sie allerdings eine ganz neue Bedeutung:

1. Wie dem aufmerksamen Leser nicht entgangen sein wird, hat die erste Definition drei und die zweite Definition nur zwei Stellen, was im Falle unserer Anwendung Sinn macht, falls das zweite E-Modul nicht bekannt ist, oder die Definitionen von zwei verschiedenen Anwendern eingeführt wurden, von denen sich der zweite nicht für E-Module interessiert. In beiden Fällen soll der Benutzer nicht gezwungen werden, die ihm unbekannten Werte durch Variablen zu substituieren. In einer nicht erweiterten RELFUN-Version hätte dies zur Folge, daß mit einer Anfrage auch jeweils nur eine Relation unifiziert werden könnte. In unsere Version können beide Definitionen unabhängig von der Anzahl der definierten Attribute mit einer Anfrage abgefragt werden.

2. In der zweiten Definition sind die Stellen von `identifier` und `density` vertauscht, was in unserem System kein Problem darstellt, da die Attributwerte in Abfragen über den Attributnamen unabhängig von der Reihenfolge identifiziert werden.

Durch diese Realisierung von Klassen mit Attribut-Wert-Paaren wird eine leichte Erweiterung der bestehenden Teile von RTPLAST ermöglicht. Insbesondere die Erweiterung ohne exakte Kenntnis wird durch ORF unterstützt; so kann ein späterer Benutzer in der KB nachschauen, ob das ihn interessierende Attribut bereits deklariert wurde. Ist dies der Fall, so muß er nur die fehlenden und ihm bekannten Attribute an beliebiger Stelle innerhalb der Definiton einfügen. Sollte das Attribut nicht deklariert sein, so muß dies nur in der entsprechenden `declare class` nachgeholt werden, wonach die Attributdefinition wie oben durchgeführt werden kann.

Da die Regeln der KB nur auf Attribute zugreifen, die innerhalb von Klassenmitgliedern definiert wurden, muß keine der bestehenden Regeln bei einer solchen Erweiterung geändert werden, da die für sie relevanten Attributwerte weiterhin über den Attributnamen und die entsprechende Klasse identifiziert werden können.

Nachdem die obigen Deklarationen und Definitionen in RELFUN geladen und compiliert wurden, können die Attributwerte der Kunststoffe im Emulatormodus wie folgt abgefragt werden:

```
rfe-p> novodur(
         identifier[novodur_r_5320],
         density[Density])
true
```

```
Density = 0.84
```

An diesem Beispiel ist zu erkennen, wie zwei der drei definierten Attribute von `novodur_r_5320` abgefragt werden. Da `novodur` als Klasse definiert wurde, muß das E-Modul nicht durch eine Variable substituiert werden, sondern wird bei der Abfrage erst gar nicht angegeben. Ebenfalls bedingt durch die Klassendefinition von `novodur` hätten die Positionen von `identifier` und `density` vertauscht werden können, ohne das sich das Ergebnis geändert hätte. Ansonsten verhält sich die obige Anfrage wie ein normales Prolog-Goal, d. h. durch die Vorgabe des konstanten Attributwertes `novodur_r_5320` für den `identifier` kann nur genau das Individuum der Klasse `novodur` unifiziert werden, in dem der Wert des `identifier` `novodur_r_5320` ist. Durch die Wahl einer Variablen[3] für den Attributwert von `density` liefert die Unifikation die Bindung der Variablen `Density` an den Attributwert des Individuums, das durch den konstanten `identifier` schon festgelegt wurde. Der Vorgang, der hier stattfindet, ist also nichts anderes als Unifikation[4] auf einer Menge von Individuen, die durch eine variabel lange und nicht-positionale Menge von Attribut-Wert-Paaren beschrieben werden. Mit diesen Klassen sind also einige der Forderungen bzgl. der leichten Erweiterung unserer KB aus Kapitel 2 erfüllt.

Natürlich sollen die Kunststoffdaten der KB nicht nur als Top-Level-Goal abfragbar sein, vielmehr soll jede RELFUN-Klausel in der gleichen variablen Weise auf die Daten zugreifen können, da bei Erweiterungen der KB auch die bestehenden Regeln nicht geändert werden sollen. Deshalb kann das obige Goal auch in gleicher Form in jedem Klauselrumpf einer RELFUN-Regel benutzt werden. Eine Regel, die z. B. alle Novodur-Individuen ermittelt, deren Dichte zwischen 0.8 und 1 liegt, hätte folgende Gestalt:

```
interv-density(Identifier, Density) :-
  novodur(identifier[Identifier], density[Density]),
  >=(Density, 0.8),
  <=(Density, 1).
```

Wird diese Klausel zu den obigen Klassendefinitionen geladen, so kann die Klausel direkt als Goal gestellt werden. Durch Nachfordern werden dann alle entsprechenden Novodur-Individuen auf die folgende Weise ermittelt:

[3] Variablennamen beginnen in der Prolog-Notation von RELFUN mit einem Großbuchstaben wie in Prolog selbst.

[4] Hierfür wurde nicht die Unifikation in RELFUN selbst geändert, vielmehr wird die variable Darstellung in eine positionale überführt, wodurch auch die Anfragen vor der Unifikation in eine positionale Form gebracht werden müssen.

```
rfe-p> interv-density(Identifier, Density)
true
Identifier = novodur_r_5320
Density = 0.84
rfe-p> more
true
Identifier = novodur_r_5322
Density = 0.91
rfe-p> more
unknown
```

3.2.2.3 Erlaubte Attributwerte

Nachdem im vorhergehenden Abschnitt gezeigt wurde, wie die Objekte der Kunststoffdomäne durch eine Menge von Attribut-Wert-Paaren beschrieben werden, soll in diesem Abschnitt gezeigt werden, welche Attributwerttypen in unserer Darstellung zulässig sind und wie sie verarbeitet werden können. Hierbei beziehe ich mich vor allem auf die drei in Abschnitt 3.1 beschriebenen Attributarten.

Atomare Werte Hiermit sind Werte wie der `Real`-Wert für die Dichte oder das konstante Symbol für den Identifier gemeint. Würden nur atomare Werte für die Erstellung von RTPLAST benutzt, so wäre die positionalisierte Form (siehe Abschnitt 3.3) DATALOG-gerecht bzw. in der ersten Normalenform (siehe [Ull88]), falls die Relationen als relationale Datenbank aufgefaßt würden. Da man sich bei komplexen Problemen durch eine solche Einschränkung allerdings einige Umstände bereitet,[5] enthält unsere Darstellung auch Strukturen, die für die evolutionsgerechte Sprache eine zulässige Erweiterung von DATALOG sind.

Variablen Variablen werden jedem in einer Klasse deklarierten, aber nicht definierten, Attribut automatisch zugewiesen. Dies entspricht allerdings nicht dem eigentlichen Sinn der Möglichkeit, für einige Attribute keinen Wert zu definieren, da solche 'null values' eigentlich von Variablen zu unterscheiden sind. Solche Attribute sollten also z. B. automatisch mit dem Wert `unknown` initialisiert werden. Leider war diese Erweiterung für den ORF-Prototypen zu aufwendig, so daß Variablen kaum bewußt als Modellierungsmöglichkeit genutzt werden können. Sollte RTPLAST weiter in der ORF-Version gepflegt werden, so ist die gerade beschriebene Erweiterung sicherlich sinnvoll.

Eine Möglichkeit Variablen trotzdem nützlich einzusetzen ist folgende: In manchen Anwendungen kann es vorkommen, daß verschiedene Attributwerte eines Stoffes voneinander abhängig sind. Entweder sind sie identisch, oder ein Wert kann immer

[5] Auch für die relationalen Datenbanken hat man dies erkannt, so wird der kommende SQL3-Standard auch die Abfrage von komplexen Strukturen standardisieren.

aus einem oder mehreren anderen berechnet werden. Um dies in ORF abzubilden, können *Koreferenzvariablen* benutzt werden. Für unser obiges Beispiel könnte dies wie folgt aussehen:

```
novodur(
  identifier[novodur_r_5320],
  density[X],
  tension_module_of_elasticity[X]]).
```

In diesem Beispiel wird davon ausgegangen, daß das E-Modul und die Dichte den gleichen Wert haben. Theoretisch könnte das E-Modul auch von mehreren Attributen abhängen, in diesem Fall müßten nur mehr Koreferenzvariablen eingeführt werden.

Tupel und Strukturen Tupel und Strukturen sind zulässige Attributwerte, falls sie auch zulässige RELFUN-Ausdrücke sind. Tupel werden durch eckige Klammern gekennzeichnet (z. B. `[abs, pp]`) und sind passive unbenannte Strukturen, die Lisp-Listen entsprechen. Für die Tupel sind in RELFUN verschiedene Builtins definiert [BEH+93].

Strukturen sind ebenfalls passiv und werden, anders als in Prolog, durch eckige Klammern mit einem vorangestelltem Konstruktor gekennzeichnet (z. B. `const[abs, pp]`). Strukturen können vor allem zur Strukturierung und Steigerung der Übersichtlichkeit verwendet werden. Ein Beispiel hierfür wäre das Zusammenfassen mehrerer zusammenhängender Attributwerte zu einer Struktur, was in der KB wie folgt aussehen könnte:

```
Aus:

novodur(
  identifier[novodur_r_5320],
  dimensional_stability_hdt/a[90],
  dimensional_stability_hdt/b[95]).

wird:

novodur(
  idenitifier[novodur_r_5320],
  dimensional_stability_hdt[a/b[90, 95]]).
```

Der Attributwert für `dimensional_stability_hdt` wäre im zweiten Fall die Struktur `a/b[90, 95]`. Die so erzeugten Strukturen können beliebig tief verschachtelt werden.

Methoden In manchen Fällen kann es nützlich sein, einem Attribut keinen festen Wert zuzuweisen, sondern den Wert erst zum Zeitpunkt der Anfrage zu berechnen. Funktionen, die eine solche Berechnung durchführen, werden in objektorientierten Systemen häufig Methoden[6] genannt, weshalb ich diesen Begriff ebenfalls verwenden werde. Der Einsatz solcher Methoden ist vor allem dann sinnvoll, wenn ein Attributwert von einem nicht konstantem Parameter abhängt.

An dieser Stelle muß allerdings festgestellt werden, daß die Beschreibung eines Attributs durch eine Methode für Zwecke der Evolution äußerst ungeeiget ist. Um die mit einer Methode zu ermittelnden Attribute vergleichen zu können, reicht keine Syntax- oder Strukturanalyse der sie beschreibenden Regeln. Stellt die Methode z. B. eine mathematische Formel dar, müßten mehrere Ableitungen der Funktion ermittelt werden, um deren Verlauf mit anderen Funktionen vergleichen zu können. Ein anderer Weg wäre die Berechnung meherer Stützstellen der zu vergleichenden Funktionen mit den Methoden, die dann miteinander vergleichbar wären. Beide Wege können aber sehr komplex werden und sind wohl auf großen KBs nicht anwendbar, oder zumindest erst in einem späteren Stadium der VEGA-Forschung realisierbar, weshalb beim Aufbau von RTPLAST der Einsatz von Methoden möglichst vermieden wurde.

Eine Ausnahme stellen die praxisrelevanten zweidimensionalen Diagramme dar. Da in Datenblättern über Kunststoffe bestimmte Eigenschaften immer mit solchen Diagrammen beschrieben werden, ließ sich deren Integrierung in die KB nicht vermeiden. Außerdem wurde eine Darstellung für die Diagramme gefunden, die auch zur Evolution verwendet werden kann. Hierfür wird eine Menge von das Diagramm beschreibenden Stützstellen als Attributwert gespeichert, aus denen dann der exakte Wert für die gefragte Koordinate interpoliert werden kann:

```
novodur(
  identifier[novodur_r_5320],
  stress-strain[[interpol, [[0.71, 13.4],
                    [1.42, 22.2],
                    [2.13, 26.7],
                    [2.84, 29.5],
                    [3.55, 31.6],
                    [4.26, 33.2],
                    [4.97, 34.3],
                    [5.68, 35.2],
                    [6.39, 35.7],
                    [7.1, 36.0]]]]).
```

[6] Ich will an dieser Stelle nicht den Eindruck erwecken, daß für ORF ein Methodenkonzept konzipiert wurde, welches ebenso mächtig ist wie die Konzepte herkömmlicher objektorientierter Systeme mit Dämonen, Messages usw. Vielmehr ist das ORF-Konzept sehr einfach und verlangt viel Unterstützung seitens des Benutzers.

Der Leser kann erkennen, wie das `stress-strain`-Diagramm (Spannungsdehnungsdiagramm) als Liste (Tupel) dargestellt wird, wobei das erste Element des Tupels angibt, daß die folgende Tabelle mit der Methode `interpol` verarbeitet wird, die eine Realisierung des Lagrange-Algorithmus zur Interpolation in RELFUN ist (siehe Anhang A.1). Die Tabelle selbst ist wiederum ein Tupel von zweistelligen Tupeln, welche die Stützstellen repräsentieren. Diese Tabellen können bei der Evolution relativ einfach miteinander verglichen werden.

Die Methode `interpol` wird bei der Anfrage von `stress-strain` allerdings nicht automatisch aufgerufen, wie dies in objektorientierten Systemen gemacht würde. Der Benutzer muß den gewünschten Wert wie folgt berechnen:[7]

```
rfe-p> novodur(identifier[novodur_r_5320],
               stress-strain[[Method, Chart]]),
       Method(Chart,2.3)
27.473159400564154
Method = interpol
Chart = [[0.71, 13.4],
         [1.42, 22.2],
         [2.13, 26.7],
         [2.84, 29.5],
         [3.55, 31.6],
         [4.26, 33.2],
         [4.97, 34.3],
         [5.68, 35.2],
         [6.39, 35.7],
         [7.1, 36.0]]
```

In diesem Beispiel wird für die `stress-strain`-Funktion an der Stelle 2.3 der Wert 27.47 interpoliert. Natürlich muß der Benutzer bei dieser Art der Verarbeitung die Struktur des Attributs kennen, was zwar die Wiederverwendung etwas erschwert, sich aber wohl nicht vermeiden läßt. Um solche Informationen über die Attribute in die KB aufzunehmen, enthält RTPLAST in Anlehnung an ein Datenbankschema ein Wissensbankschema, das Teil der KB ist und in Abschnitt 3.5 beschrieben wird.

Regeln Ähnlich den Methoden kann ein Attributwert mit einer Regel bestimmt werden. Regeln werden in den Fällen benutzt, in denen der Attributwert wiederum von einem bestimmten Parameter abhängt, der resultierende Wert allerdings nicht

[7] Eine automatische Berechnung von Methoden ließe sich wohl theoretisch im RELFUN-System realisieren, wäre aber mit einem großen Aufwand verbunden, der durch den vermeintlichen Gewinn wohl nicht aufgewogen wird. Außerdem wäre die automatische Methode weniger deklarativ, da das abgebildete Wissen erst nach der Verarbeitung sichtbar wird.

berechnet werden kann, sondern aus einer Menge von Alternativen ausgewählt wird. In der ORF-Notation wird in solchen Fällen lediglich der Name der Regel dem Attribut zugewiesen. Die Regel selbst wird als normale RELFUN-Regel notiert, wobei natürlich spezielle ORF-Anfragen benutzt werden. Um dies an einem Beispiel zu verdeutlichen, wird die Definition von `novodur` wie folgt erweitert:

```
novodur(
  identifier[novodur_r_5320],
  recyclable[recyclable-rule],
  additives[[flamability-trigger]]).
```

Man kann erkennen, wie `recyclable` und `additives` als Attribute mit den Werten `recyclable-rule` und dem Tupel `[flamability-trigger]` der `novodur`-Defintion hinzugefügt werden (vorher müßten `recyclable` und `additives` natürlich deklariert werden). Mit `additives` wird angegeben, welche Zusätze der Kunststoff enthält, wobei mit `flamability-trigger` Flammhemmer gemeint sind. Mit dem Attribut `recyclable` soll ausgedrückt werden, inwieweit ein Kunststoff recyclierbar ist. Beschreibt man dieses Attribut nur qualitativ, z. B. mit 'gut' oder 'schlecht', so ist diese Information für eine realistische Anwendung wohl nicht speziell genug; deshalb habe ich z. B. mit der folgenden Regel in dem Prototypen versucht, den Begriff 'recyclierbar' etwas genauer zu beschreiben [8]:

```
recyclable-rule(Plastic-id) :-
  instance>(absc(), Class(identifier[Plastic-id],
                          additives[Additives])),
  nonvar(Additives),
  member(flamability-trigger, Additives) &
  only_in_closed_circle.
```

Dies ist eine der definierten `recyclable-rule` Regeln. Sie benutzt bis auf den `instance>`-Operator nur in RELFUN zulässige Prädikate und Funktionen. Die genaue Arbeitsweise des `instance>`-Operators wird in Abschnitt 3.2.2.5 erklärt. An dieser Stelle soll nur beschrieben werden, wie mit ihm in der Vererbungshierarchie die Liste der Additive des Kunststoffs gefunden wird, dessen Identifier im Regelkopf an `Plastic-id` gebunden wurde. Sollte der Kunststoff kein `absc` sein (siehe Abbildung 3.1), so 'failed' der Operator und damit die ganze Regel, wodurch ausgedrückt wird, daß diese spezielle Definition nur für die ABSC-Kunststoffe

[8] Wie diese Eigenschaft mit Hilfe einer Kennziffer sehr genau differenziert werden kann, wird in [Kre94] beschrieben.

gilt. Ist nun dieser Kunststoff gefunden und ist die Liste seiner Additive definiert (`nonvar(Additives)`), so wird geprüft, ob Flammhemmer in der Liste der Additive enthalten sind (`member(flamability-trigger, Additives)`). Ist dies der Fall, so wird `only_in_closed_circle` als Ergebnis zurückgegeben, was in RELFUN durch das vorangestellte `&` gekennzeichnet wird.

Die Bedeutung dieser Regel ist folgende: ABSC-Kunststoffe sind in der Regel zwar recyclierbar, doch wenn sie Flammhemmer enthalten, können sie nur innerhalb geschlossener Kreisläufe recycliert werden, da die Information über das Enthaltensein der Flammhemmer nicht im Recyclierungsprozeß verloren gehen darf, weil sonst bei einer späteren unsachgemäßen Verbrennung der Kunststoffe Dioxin entstehen kann.

Die Verarbeitung der Regeln muß wie bei den Methoden von Hand erfolgen. Für die obige Regel sieht das wie folgt aus:

```
rfe-p> novodur(identifier[Identifier], recyclable[Rule]),
       Rule(Identifier)
only_in_closed_circle
Identifier = novodur_r_5320
Rule = recyclable-rule
```

Dieses Goal würde durch Nachfordern natürlich alle bekannten `recyclable`-Eigenschaften der `novodur`-Individuen liefern. Die Struktur der Regel-Attribute wird ebenfalls wie bei den Methoden im Wissensbankschema beschrieben.

3.2.2.4 Vererbung durch Prototyp-Klassen

Nachdem mit Hilfe der Individual-Klassen die unterste Ebene der Kunststoffdomäne beschrieben werden kann, wird nun gezeigt, wie mit den sogenannten **Prototyp-Klassen** die Domäne strukturiert und gleichzeitig die Forderung nach geringer Redundanz durch Vererbung erfüllt wird.

Vor Auswahl der Repräsentationssprache für RTPLAST stand fest, daß die Sprache Möglichkeiten zur Strukturierung enthalten sollte. Dies war zum einen dadurch bedingt, daß solche Strukturierungen der Domäne bereits in der Literatur vorhanden waren [KLW93] und in dieser Form auch als Domänenwissen in der KB enthalten sein sollten. Zum anderen war die Vermeidung von Redundanz eine Anforderung an die KB, die mit Hilfe von Vererbungsmechanismen zu erfüllen war. Es ist klar, daß die Vererbung nur entlang einer die Domäne beschreibenden Hierarchie erfolgen kann. Hier konnte also eine Lösung für zwei Probleme gefunden werden.

Eine Klasse von Sprachen, die zum einen diese Darstellungen unterstützen und zum anderen auch deklarativ sind, ist die Klasse der terminologischen Systeme, deren Ursprung KL-ONE [BS85] ist. Die Untersuchung auf die Anwendbarkeit solcher Sprachen für unsere Domäne ergab allerdings, daß die angebotenen Darstellungsmöglichkeiten für unser Problem zu mächtig waren. Dies ist ein Problem, da für die Evolution

ein möglichst kleiner Sprachkern zu wählen ist. So konnten zwischen den Konzepten/Klassen unserer Domäne keine interessanten Rollen gefunden werden, also auch keine, die mit einem Existenz- oder Allquantor speziellere Konzepte hätten beschreiben können. Vielmehr läßt sich die Kunststoffdomäne allein dadurch gliedern, daß bestimmte Kunststoffgruppen teilweise gleiche Attributwerte haben. Zur Beschreibung dieser Gruppen reicht also ein einfaches, Vererbung unterstützendes Klassensystem, welches als Erweiterung von RELFUN prototypisch implementiert wurde. Außerdem sprach das Fehlen von effizienten Schnittstellen zwischen terminologischen und regelverarbeitenden Komponenten gegen die Benutzung eines terminologischen Systems zur Strukturierung der Domäne. Auch der von den terminologischen Systemen als Hauptdienst angebotene Classifier wäre für die RTPLAST-Evolution keine Hilfe, da die Kunststoffdomäne bereits hinreichend durch Experten strukturiert wurde (siehe [KLW93]).

Der Kerngedanke der terminologischen Systeme, nämlich die Strukturierung der Domäne in Konzepte, die in einer Hierarchie angeordnet werden, ist im Prinzip sinnvoll, und sollte von uns unterstützt werden. Dies kann in der evolutionsgerechten Sprache durch die geplante Erweiterung um Sortenhierarchien realisiert werden, da die sortierte Hornlogik genau diesem Kern der terminologischen Sprachen entspricht. Ein sehr wichtiger Unterschied der sortierten Hornlogik zu terminologischen Sprachen wäre dabei, daß die Abarbeitung der Sorten in die Unifikation eingebaut ist, somit keine Schnittstelle zwischen der Sortendarstellung und der Regeldarstellung gebraucht wird. Wie das Klassenkonzept in die sortierte Hornlogik übertragen werden kann, wird in Kapitel 4 beschrieben.

Zur Realisierung der geforderten Vererbung von Attributwerten an die Individuen werden Prototyp-Klassen benutzt. Um die Verwendung der Prototyp-Klassen am bisherigen Beispiel erklären zu können, wird die Liste der Deklarationen wie folgt erweitert:

```
declare(proto-class[absc,
                    super[thermoplastic],
                    ball_thrust_hardness, additives]).

declare(indi-class[novodur,
                   super[absc],
                   identifier, density,
                   tension_module_of_elasticity]).
```

Es ist zu erkennen, daß sich die Deklarationen der Prototyp-Klassen nur durch die Substitution von `indi` durch `proto` unterscheiden. Durch die in `super` definierten Klassen wird hier eine Hierarchie aufgebaut, wobei die hierdurch entstehende Relation im Prinzip der `is-a`-Relation aus objektorientierten Systemen entspricht. Die Bedeutung der neuen Attribute ist folgende:

`ball_thrust_hardness`: gibt die Eindrucktiefe einer mit einem normierten Druck auf den Kunststoff gepreßten Kugel an.

`additives`: gibt die Liste der in einem Thermoplast enthaltenen Additive (Zusätze) an.

Für unser Beispiel gilt nun, daß die für diese beiden Attribute zu definierenden Werte für alle Individuen der Klasse `novodur` gleich sind, und somit von `absc` vererbt werden können.

Die Definition dieser neuen Attribute erfolgt ebenfalls wie bei den Individual-Klassen:

```
absc(
  ball_thrust_hardness[90],
  additives[[]]).
```

Der einzige Unterschied zur Definition der Individuen besteht darin, daß für diese Klassen kein Identifier definiert wird, was auch keinen Sinn ergeben würde, da sie ja lediglich eine Ansammlung von zu vererbenden Werten darstellen. In der Regel wird für jede Prototyp-Klasse auch nur eine Definition notiert. Weitere Definitionen sind bei dieser nur die Vererbung ausnutzenden Verwendung von ORF sinnlos, da dadurch alle in der Vererbungskette liegenden Individuen zweimal instanziert wären. Der Leser mag sich dies nach der Lektüre von Abschnitt 3.3 noch einmal verdeutlichen.

Sollten die in `absc` deklarierten Attribute nicht in `absc` definiert werden, so können sie als zulässige Attribute auch erst in `novodur` definiert werden. Ein Überschreiben der in `absc` definierten Attributwerte ist in tiefer liegenden Klassen nicht möglich, da während der Vererbung versucht würde, die beiden Werte zu unifizieren.[9] Wie diese evtl. sinnvolle Erweiterung implementiert werden kann, wird in Kapitel 4 erläutert.

Die hier definierten Werte können nun in den Klassen `absc` oder `novodur` wie bisher abgefragt werden:

```
rfe-p> absc(ball_thrust_hardness[Ball])
true
Ball = 90
rfe-p> more
unknown
rfe-p> novodur(identifier[Id], ball_thrust_hardness[Ball])
true
```

[9] Siehe Abschnitt 3.3.2.

```
Id = novodur_r_5320
Ball = 90
rfe-p> more
true
Id = novodur_r_5322
Ball = 90
rfe-p> more
unknown
```

3.2.2.5 Hierarchische Suche

Mit den oben eingeführten Konstrukten kann die Kunststoffdomäne in einer objektzentrierten, hierarchisch angeordneten und Vererbung unterstützenden Weise modelliert werden; wie dies aussieht, kann in Anhang A.1 nachgelesen werden. Will der Benutzer in der bisherigen Form Kunststoffe anhand eines Anforderungsprofils[10] auswählen, so muß er für jede Individual-Klasse prüfen, ob in ihr entsprechende Individuen enthalten sind. Dies erfordert nicht nur eine zu vermeidende präzise Kenntnis der Struktur der KB, sondern ist für die praktische Auswahl auch viel zu aufwendig. Daher enthält ORF Konstrukte zur Abfrage bestimmter Teile der Kunststoffhierarchie.

Um die Funktionsweise dieser Konstrukte erläutern zu können, wird die typische Materialauswahl an dieser Stelle kurz erklärt: Bei der Konstruktion eines technischen Bauteils wird wegen allgemeiner Anforderungen zuerst eine Klasse von Materialien festgelegt, aus der das Produkt erstellt werden soll. Diese erste grobe Auswahl kann mehr oder weniger speziell sein, z. B. kann die erste Analyse des Konstruktionsproblems ergeben, daß für das Produkt alle Thermoplaste oder spezieller nur Polypropylene in Frage kommen. Nachdem diese erste grobe Auswahl durchgeführt wurde, wird der gesuchte Kunststoff durch die Festlegung oder Einschränkung seiner Attributwerte näher spezifiziert. Anhand dieser Spezifikation wird dann eine Menge von konkreten, verwendbaren Kunststoffen ausgewählt. Für die Suche in einer ORF-Hierarchie bedeutet dies, daß der Benutzer die Möglichkeit haben muß, alle Attribute von Individuen, die unterhalb einer bestimmten Klasse liegen, mit den geforderten Attributwerten zu vergleichen.

In ORF heißt die für diese Aufgabe bereitgestellte Funktion `instance>`, die anhand des folgenden Beispiels erklärt werden soll, wobei die Annahme gelte, daß die Klassen `thermoplastic`, `absc` und `novodur` mit den Attributen `identifier`, `density`, `tension_module_of_elasticity`, `ball_thrust_hardness` und `additives` aus Abbildung 3.1 bereits deklariert und definiert sind:

[10] Im Anforderungsprofil werden bestimmte, vom gesuchten Kunststoff zu erfüllende Eigenschaften beschrieben, was hauptsächlich durch die Einschränkung oder Festlegung von Attributwerten realisiert wird.

```
rfe-p> instance>(thermoplastic(),
                 Class(identifier[Plastic-id],
                       tension_module_of_elasticity[E-module],
                       density[0.84])),
       >(E-module, 1700)
true
Class = novodur
Plastic-id = novodur_r_5320
E-module = 2000

rfe-p> more
unknown
```

An diesem Beispiel ist zu erkennen, wie durch den `instance>`-Operator eine Menge von Attributwerten einer unterhalb der Prototyp-Klasse `thermoplastic` liegenden Individual-Klasse ermittelt werden. Die Angabe der Prototyp-Klasse, ab der gesucht werden soll, steht hierbei an der ersten Stelle der `instance>`-Funktion. Die zweite Stelle enthält zunächst eine Variable (`Class`), an die im Laufe der Unifikation der Klassenname gebunden wird, dessen Individuum gerade unifiziert wird. In den Klammern folgt hierauf die genaue Spezifikation der gesuchten Attribute, die wie gewohnt notiert werden und ebenfalls wie bisher in variabler Reihenfolge und Auswahl hingeschrieben werden können. Durch die Angabe einer Konstanten für den Attributwert kann nun die Auswahl so gesteuert werden, daß nur solche Kunststoffe gewählt werden, die für dieses Attribut genau diesen Wert haben (`density`). Soll der Wertebereich eines Attrributs nur eingeschränkt werden, so ist für den Attributwert eine Variable zu wählen (`tension_module_of_elasticity`), die durch ein mit dem `instance>`-Operator konjunktiv verknüpftes RELFUN-Builtin eingeschränkt wird (`>(E-module, 1700)`). Alle weiteren interessanten Werte sind ebenfalls an Variablen zu binden (`identifier`).

Der `instance>`-Operator kann nicht nur wie oben als Top-Level-Goal verwendet werden, sondern ist natürlich in gleicher Weise in jeder RELFUN-Klausel zu benutzen. Ein Beispiel hierfür findet der Leser in dem Paragraphen über durch Regeln spezifizierte Attributwerte (`recyclable-rule`).

Da es nicht immer sinnvoll ist, gleich nach Individuen zu suchen, bietet ORF auch Möglichkeiten zur Suche innerhalb beliebiger Klassen. Die entsprechenden Operatoren heißen `class>` und `class>=` und unterscheiden sich dadurch, daß `class>=` im Gegensatz zu `class>` die Klasse, ab der gesucht werden soll, bei der Suche einschließt. Die Syntax der beiden Operatoren entspricht der von `instance>` (und `instance>=`). Wird bei der Suche mit diesen beiden Operatoren die Definition von Attributwerten entdeckt, welche die gewünschten Anforderungen erfüllen, so ist dies so zu interpretieren, daß alle unterhalb der gefundenen Klasse definierten Individuen ebenfalls diese

Anforderungen erfüllen. In unserem bisherigen Beispiel könnte entdeckt werden, daß alle Individuen unterhalb der Klasse `absc` gewählt werden können, falls z. B. nur die `ball_thrust_hardness`-Werte relevant sind.

3.3 Die Überführung von ORF nach RELFUN

Durch die oben eingeführten Konstrukte erfüllt RTPLAST zwar die von der Anwenderseite kommenden Ansprüche, allerdings ist sie wohl nicht mehr sehr deklarativ und kann aufgrund der nur partiellen Definition der Objekte durch Attribut-Wert-Paare auch nur zum Testen von sehr speziellen Evolutions-Algorithmen benutzt werden. Daher war ein Ziel dieser Diplomarbeit, die KB so zu gestalten, daß sie möglichst automatisch in die sich entwickelnde VEGA-Sprache[11] überführt werden kann. Zur einfachen Überführung der objektzentrierten Darstellung in eine logiknahe bietet sich folgende Positionalisierung an: *Erzeuge aus einem mit n Attributen beschriebenen Objekt eine n-stellige Relationen, wobei jede Stelle der Relation einem Attributwert entspricht.* Die positionale Darstellung hätte weiterhin den Vorteil, daß sie direkt in RELFUN dargestellt und verarbeitet werden kann. Es lag also nahe, ORF als Transformation (Vorverarbeitung) zu implementieren, deren Ausgabe direkt im RELFUN-System verarbeitet werden kann. Durch diese Vorverarbeitung der ORF-Konstrukte wurde also eine einfache, prototypische ORF-Implementierung möglich, für die keine Erweiterung der eigentlichen Unifikation von RELFUN erforderlich war.

3.3.1 Darstellung von Objekten als Relation

Um eine Definition einer Individual-Klasse als Relation beschreiben zu können, werden die ungeordnet definierten Attribute intern in eine festgelegte Reihenfolge gebracht. Hierfür wird aus der Deklaration der Individual-Klasse eine kanonisch (alphabetisch) geordnete Liste der deklarierten Attribute erzeugt. Jede Definition eines Individuums dieser Klasse wird nun in eine Relation umgeformt, deren Stelligkeit der Anzahl der Attribute entspricht. Jeder definierte Attributwert wird an entsprechender Stelle in der Relation eingefügt und die restlichen Stellen werden mit verschiedenen Variablen[12] aufgefüllt.

Durch Laden und Compilieren der ORF-KB in RELFUN wird diese in die positionalisierte Form gebracht, deren Listing für die bereits bekannten Deklaration und Definitionen wie folgt aussieht:

[11] Die Grundlagen der VEGA-Sprache wurden bereits in Kapitel 2 beschrieben.

[12] Die Problematik, die durch die Auffüllung mit Variablen entsteht, wurde bereits in Abschnitt 3.2.2.3 erläutert.

Deklaration:

```
declare(indi-class[novodur,
                   super[absc],
                   identifier, density,
                   tension_module_of_elasticity]).
```

Definition (Attributlisten):

```
novodur(
  identifier[novodur_r_5320],
  density[0.84],
  tension_module_of_elasticity[2000]).

novodur(
  density[0.91],
  identifier[novodur_r_5322]).
```

Definition (Positionalform):

```
novodur(0.84,
        novodur_r_5320,
        2000).

novodur(0.91,
        novodur_r_5322,
        Tension_module_of_elasticity8).
```

Bedingt durch die Überführung der Klassen in Relationen müssen alle Top-Level-Goals, die sich auf ORF-Klassen beziehen, vor ihrer Abarbeitung ebenfalls in die positionale Form überführt werden, wofür wiederum die intern gehaltenen Listen der Klassen und ihrer Attribute benutzt werden. Diese Überführung ist in den RELFUN-Emulator integriert, der sie vor Abarbeitung des Top-Level-Goals automatisch durchführt. Obwohl diese Überführung auch in den Interpreter eingebaut werden könnte, sind ORF-KBs zur Zeit nur im Emulatormodus zu benutzen.

Die zu RTPLAST gehörenden RELFUN-Klauseln, die sich auf ORF-Klassen beziehen,[13] werden während der Compilation in die positionalisierte Form gebracht.

[13] Ein Beispiel hierfür ist die bereits beschriebene `recyclable-rule`, die den `instance>`-Operator enthält.

Hierfür werden die ORF-Konstrukte in den Klauseln durch Relationen ersetzt, wobei die bisher durch den Attributnamen gekennzeichneten Variablen und Konstanten an der entsprechenden Stelle der Relation plaziert, und die restlichen Stellen der Relation mit Variablen gefüllt werden. Somit kann eine überführte ORF-KB auch in einer nicht erweiterten RELFUN-Version verwendet und mit normalen RELFUN-KBs gemischt werden, falls alle Top-Level-Goals nur direkt auf die Relationen (und nicht auf Klassen) zugreifen.

3.3.1.1 Erzeugen von positionalisierten ORF-Wissensbasen

Es sei an dieser Stelle daran erinnert, daß Evolutions-Algorithmen so entwickelt werden sollten, daß sie auf möglichst viele KBs angewendet werden können. Hierfür ist es erforderlich, für diese KBs eine einheitliche Repräsentation oder zumindest eine einheitliche Basis, mit deren Hilfe die unterschiedlichen Repräsentationen ineinander überführt werden können, zu finden. Da bereits Arbeiten zur Entwicklung solcher Wissensaustauschformate existieren (z. B. KIF, ML^2) [Rah93], sollte die Test-KB der Evolutions-Algorithmen in einer möglichst einfach in diese Standards zu überführenden Repräsentation existieren. Da diese Standards logikbasiert sind, sollte RTPLAST auch in einer logiknahen Version instanziert sein.

Durch die Verwendung der partiell definierten Attribut-Wert-Paare und Klassen ist ORF für diese Aufgabe nicht gut geeignet. Jedoch kann die positionalisierte Form von RTPLAST hierfür verwendet werden, da sie das enthaltene Wissen in hornlogische Fakten und Regeln abbildet.

Zur Erzeugung einer aktuellen positionalisierten Form von RTPLAST ist diese zuerst in das RELFUN-System zu laden. Anschließend sind die Kommandos `untype` und `normalize` auszuführen, die die positionalisierte Form erzeugen und gewöhnlich in den Compiler integriert sind. Nun kann mit dem Kommando `tell` die positionalisierte KB in eine Datei geschrieben werden. Durch diese automatische Überführung kann RTPLAST weiterhin in der benutzerfreundlichen und leicht erweiterbaren ORF-Version gepflegt und verwendet werden. Ob dies als ständige Vorverarbeitung Sinn macht, ist allerdings noch zu klären, da dann z. B. Methoden gefunden werden müßten, um die Validierungs- und vor allem Explorationsergebnisse auf der positionalen Form in die ORF-Form zurückzutransformieren.

3.3.1.2 Nachteile der positionalen Darstellung

Welche Probleme bei der positionalen Darstellung durch eine große Anzahl von Attributen entstehen, wurde bereits an mehreren Stellen dieser Arbeit beschrieben (siehe Abschnitte 3.2.2.1, 3.2.2.2, 2.1.3.3), so daß dies hier nur noch einmal erwähnt und nicht erläutert werden soll.

Ein weiterer Nachteil ist der Verlust der Attributbezeichner während der Übertragung, da nur die Attributwerte in die Relationen aufgenommen werden. Daher werden die Attribute in der positionalen Version im Gegensatz zu ORF nicht KB-intern

beschrieben, sondern können nur in einem externen Schema repräsentiert werden; d. h., jeder Benutzer der KB benötigt eine Beschreibung, in der er nachlesen kann, an welcher Stelle einer Relation er das von ihm gesuchte Attribut findet. Sollte die KB dann in der ORF-Version verändert werden, so müßte stets auch die Beschreibung angepaßt werden. Da im RELFUN-System zur Laufzeit zu jeder ORF-Klasse stets die alphabetisch geordnete Liste ihrer Attribute gehalten wird, könnte diese in eine Datei oder als Kommentar in die KB selbst geschrieben und so für den Benutzer zugänglich gemacht werden.

Die Nachteile der ORF- und der positionalen Darstellung waren der Anlaß zur Entwicklung einer Darstellung, die die Vorteile beider Sprachen vereint, deren Nachteile aber weitgehend vermeidet. Diese Repräsentation wird in Kapitel 4 beschrieben.

3.3.2 Vererbung mit Relationen

Bisher wurde nur beschrieben, wie ein Individuum einer einzelnen Klasse in eine Relation überführt wird. Wie die in ORF realisierte Vererbung in dieses Konzept zu integrieren ist, wird in diesem Abschnitt beschrieben.

Das Grundprinzip der hier realisierten Vererbung ist einfach und soll am folgenden bekannten Beispiel erläutert werden:

Erweiterung der Deklaration:

```
declare(proto-class[absc,
                    super[thermoplastic],
                    ball_thrust_hardness, additives]).
```

Erweiterung der Definition (Attributliste):

```
absc(
  ball_thrust_hardness[90],
  additives[[]]).
```

Erweiterung der Definition (Positionalform):

```
thermoplastic(...).

absc(90,
       [])
    :- thermoplastic(...).
```

```
novodur(0.84,
        novodur_r_5320,
        2000,
        Ball_thrust_hardness5,
        Additives6)
    :- absc(Ball_thrust_hardness5,
            Additives6).

novodur(0.91,
        novodur_r_5322,
        Tension_module_of_elasticity8,
        Ball_thrust_hardness12,
        Additives13)
    :- absc(Ball_thrust_hardness12,
            Additives13).
```

Für jede deklarierte und definierte Klasse wird auf die bereits bekannte Weise eine Relation erzeugt. Besteht nun aufgrund der Klassendeklaration eine direkte Vererbungsbeziehung zwischen zwei Klassen, so wird eine RELFUN-Regel generiert, wobei die als `super` deklarierte Klasse den Rumpf (`absc`) und die Unterklasse den Kopf (`novodur`) bildet. Die Vererbung erfolgt nun durch die Gleichbenennung der entsprechenden Variablen im Rumpf und im Kopf.

Diese Realisierung hat folgende Konsequenzen:

1. Auch wenn keine Werte definiert werden, muß jede in einer Hierarchie deklarierte Klasse auch definiert werden, da sonst die Kette der Vererbungsklauseln unterbrochen wird.

2. Werden Attribute in der Hierarchie doppelt und unterschiedlich definiert, so wird bei der Abarbeitung versucht, die beiden Werte/Konstanten zu unifizieren; dies muß immer scheitern, so daß keine der Relationen dieser Kette abgefragt werden kann. Nach Meyer und Weigel [MW93] ist dies ein typisches Merkmal der 'Feature-Logiken'. Sie unterscheiden Feature-Systeme von objektorientierten Systemen durch die in Feature-Termen fehlende Möglichkeit, Attribute in der Vererbungskette nicht-monoton zu überschreiben.

3. Wird ein Prototyp mehrmals definiert, so werden nicht alle definierten Werte zusammen vererbt, sondern es ensteht für jede Prototypdefinition eine eigene Vererbungskette, wodurch die Individuen dieser Kette mehrmals definiert sind.

3.4 Eine Wissensbankinstanz von RTPLAST

RTPLAST ist der erste Protoyp einer KB über recyclierbare Thermoplaste, mit dem in dieser Arbeit erste Ergebnisse über die deklarative Repräsentation von Stoffwissen aus diesem Gebiet gewonnen wurden. Um die bzgl. der Repräsentation und Evolution getroffenen Entscheidungen überprüfen zu können, wurde eine kleine überschaubare Teilmenge der Thermoplaste repräsentiert. In einer in [KLW93] abgebildeten Kunststoffhierarchie werden die Thermoplaste in 12 Unterklassen unterteilt. Von diesen Unterklassen sind in RTPLAST einige Thermoplaste der Klassen Polypropylen (PP) und Acrylonitrile-Butadiene-Styrole-Copolymere (ABSC) repräsentiert. Die PPs wurden für den Prototypen gewählt, da:

- 95 Prozent aller GMTs aus PP hergestellt werden.
- Die PPs für das Konstruieren in Kunststoffen eine bedeutende Rolle spielen, da sie sehr kostengünstig und vielseitig zu verwenden sind.
- Bereits Verfahren zur Recyclierung von PPs existieren.

Um die PPs bereits innerhalb des Protoypen mit Thermoplasten anderer Klassen vergleichen zu können (Evolution), wurden einige ABSCs in RTPLAST hinzugefügt. Die Auswahl der ABSC-Thermoplaste für diesen Zweck basiert auf der Tatsache, daß die repräsentierten ABSCs bereits in einer garantierten Qualität als Recyclat hergestellt werden.

Welche Eigenschaften (Attribute) dieser Thermoplaste in RTPLAST konkret repräsentiert sind, kann man in Anhang A.1 der dort abgedruckten Instanz von RTPLAST entnehmen; hierbei kann das Glossar als Hilfestellung hinzugezogen werden.

3.4.1 Warum ist RTPLAST eine *Wissens*bank?

RTPLAST enthält vor allem Fakten und zur Zeit relativ wenig Regeln. Die Regeln wurden zum Teil manuell aus den (natürlichsprachlichen) Herstellerprospekten und Interviews mit Domänenexperten gewonnen; zum Teil werden sie automatisch aus der Vererbungshierarchie erzeugt. Die Fakten wurden aus der CAMPUS-Datenbank und aus (tabellarischen) Herstellerdatenblättern übernommen. Es stellt sich die Frage, warum diese zum Teil bereits gespeicherten Fakten neu formalisiert wurden und nicht über eine Datenbankschnittstelle (z. B. SQL) den Evolutionstools zugänglich gemacht werden. Dies hatte drei Gründe:

1. CAMPUS ist in einem nicht-standardisierten Format implementiert. Daher existiert keine dynamische Schnittstelle, mit der die Fakten für die Evolutionstools zugänglich geworden wären.
2. Das für die Evolution geeignete Wissen sollte in einer einheitlichen deklarativen Weise repräsentiert sein. Da Fakten ein wichtiger Spezialfall von Wissen sind, sollten sie in der gleichen Weise repräsentiert sein wie Regeln.

3. Die in CAMPUS fehlende Strukturierung der Thermoplaste [KLW93] in der Vererbungshierarchie des RTPLAST-Schemas (vgl. Abschnitt 3.5) beinhaltet bereits vorexploriertes Wissen. Dieses Wissen sollte explizit von der RTPLAST-Instanz aus zugegriffen werden können.

3.5 Das Wissensbankschema von RTPLAST

An verschiedenen Stellen dieser Arbeit wurde bereits darauf hingewiesen, daß für die Verarbeitung und Verwendung des Wissens wiederum Wissen über die Attribute, also Meta-Wissen, benötigt wird. Dies wurde für RTPLAST auf zwei verschiedene Arten realisiert:

1. Es wurde ein Schema als Teil der KB implementiert, in dem die Attribute ebenfalls zu Klassen zusammengefaßt sind, welche wiederum durch eine Menge von Attributen beschrieben werden. Dieser Ansatz wird auch für *Cyc* verwendet [LG89].

2. Zur Strukturierung der Attribute wurden diese in eigens hierfür angelegten Klassen deklariert, in denen sie nicht definiert werden.

3.5.1 Beschreibung durch Klassen

Zur Beschreibung der Attribute der KB wurde eine Hierarchie von ORF-Klassen angelegt, wobei die Individual-Klassen die konkreten Attribute beschreiben, die über den `identifier` identifiziert werden, d. h. der `identifier` der Definition einer Individual-Klasse enthält als Wert den Namen des zu beschreibenden Attributs. Um auch hier eine möglichst redundanzfreie Darstellung zu ermöglichen, wurden die entsprechenden Attribute durch Protoyp-Klassen vererbt. Mit den Definitionen dieser Klassen ist es möglich, die Attribute der KB wiederum durch eine Menge von Attributen zu beschreiben; hierbei sind zwei Arten von beschreibenden Attributen zu unterscheiden:

1. Attribute die vom System zur Evolution verwendet werden können. Hierbei handelt es sich hauptsächlich um Eigenschaften, wie sie auch in Datenbanken zur Validierung benutzt werden. Dies sind vor allem die Kardinalitäten und Sorten der Attributwerte, wobei die Sorten bei dieser Verwendung nicht die gleiche Bedeutung haben wie in der sortierten Hornlogik[14], sondern wie normale Typen zu behandeln sind, also etwa integer, real usw.

2. Attribute die dem Benutzer weitere Informationen zu dem zu beschreibenden Attribut liefern. Dies sind z. B.:

[14] Siehe Abschnitt 4.2

- ob der Attributwert erst mit einer Regel oder Methode bestimmt werden muß,
- welche physikalische Einheit die numerischen Attribute haben,
- nach welcher Norm und unter welchen Bedingungen der Attributwert ermittelt wurde.

Obwohl diese Attribute in erster Linie für den Benutzer gedacht sind, können sie natürlich auch für die Evolution genutzt werden; so kann z. B. durch die Mitführung der physikalischen Einheiten bei Berechnungen deren Ergebnis validiert werden.

Die konkreten Deklarationen und Definitionen dieser Klassen findet der Leser in Anhang A.1.

3.5.2 Deklarierende Klassen

Um die Deklaration der Attribute nicht über die gesamte KB zu verteilen, wurden spezielle Klassen angelegt, in denen die Attribute nur deklariert und nicht definiert werden. Da diese Klassen in der Hierarchie oberhalb der Kunststoffe liegen, wird die Deklaration der Attribute auch an die konkreten Klassen vererbt, in denen sie dann definiert werden. Wie die Attribute durch die deklarierenden Klassen strukturiert werden, ist Abbildung 3.1 zu entnehmen.

Durch diese Strukturierung erhält ein neuer Benutzer bei Anfragen oder Erweiterungen die einfache Möglichkeit herauszufinden, ob ein ihn interessierendes Attribut bereits in der KB deklariert wurde. Ohne diese Zusammenfassung müßte er alle Klassen auf die vorhandene Deklaration eines bestimmten Attributs prüfen, wodurch ein nicht vertretbarer Aufwand entstehen würde.

Kapitel 4

Sortierte Attribut-Wert-Paare

Nach der Fertigstellung des Prototypen von RTPLAST in der ORF-Repräsentation konnte geprüft werden, ob mit den vorhandenen Darstellungen alle Anforderungen aus Kapitel 2 erfüllt werden können. Beide Darstellungen hatten dabei die bekannten Nachteile:

- Die n-stellige positionale Darstellung ist für den Benutzer bei Anfragen und Erweiterungen zu umständlich.
- Durch die Verwendung der partiell definierten Attribut-Wert-Paare sowie die Unterscheidung zwischen Deklarationen und Definitionen[1] kann die ORF-Darstellung nur sehr eingeschränkt als Testbett der Evolutions-Algorithmen dienen.

Auch wenn es zuerst so scheint, als ob durch die automatische Überführung von ORF in die positionale Darstellung allen Ansprüchen gerecht werden kann, hat diese Vorgehensweise doch einen Nachteil: Wird die KB in ORF erstellt, gepflegt und erweitert, aber später in der positionalen Form exploriert und validiert, so stellt sich die Frage, wie die hier gewonnenen Ergebnisse in die ORF-Form zurückgespeist werden können. Dafür müßte es eine bijektive Abbildung zwischen beiden Darstellungen geben. Da ORF im Gegensatz zur positionalen Darstellung zwischen der Deklaration und Definition von Attributen unterscheidet, wäre es, wenn überhaupt, nur schwer möglich eine Funktion zu implementieren, die diese Abbildung realisiert. Es sollte also eine Möglichkeit gefunden werden, das in RTPLAST enthaltene Wissen direkt in einer evolutionsgerechten Weise zu repräsentieren, ohne dabei auf die Vorteile und Mächtigkeit der ORF-Darstellung zu verzichten. Schließlich sollte es möglich sein, diese Darstellung automatisch aus der bisherigen zu erzeugen.

Wie eine solche Darstellung aussieht, wird in den nächsten Abschnitten beschrieben, wobei zuerst erläutern wird, wie die Attribut-Wert-Paare repräsentiert werden,

[1] Um die Deklarationen und Definitionen in den richtigen Zusammenhang zu bringen, würde z. B. der Validierer spezielles Meta-Wissen benötigen, was für die Evolutions-Test-KB möglichst zu vermeiden ist.

und anschließend gezeigt wird, wie das Klassenkonzept aus ORF in die sortierte Hornlogik überführt wird. Da die meisten in den folgenden Abschnitten beschriebenen Abbildungen rein syntaktische Transformationen sind, ist eine automatische Übersetzung für die meisten ORF-Konstrukte möglich und wurde bereits implementiert.

4.1 Attribut-Wert-Paare als zweistellige Relationen

Wie in ORF wird in der neuen Darstellung ein Kunststoff durch eine Menge von Attribut-Wert-Paaren beschrieben. Zur Realisierung einer variablen Darstellung werden die Attribute eines Kunststoffs allerdings nicht mehr zu einer Klasse zusammengefaßt, sondern werden als zweistellige Relation notiert, wobei der Bezeichner der Relation dem Attributnamen entspricht, weshalb diese Darstellung ab jetzt **attributzentriert** genannt wird. Die folgende Umwandlung soll dies verdeutlichen:[2]

ORF:

```
novodur(
  identifier[novodur_r_5320],
  density[0.84],
  tension_module_of_elasticity[2000]).
```

Attributzentriert:

```
density(novodur_r_5320, 0.84).
tension_module_of_elasticity(novodur_r_5320, 2000).
```

An diesem Beispiel ist zu erkennen, wie der in ORF als normales Attribut eingeführte `identifier` nun eine ausgezeichnete Rolle bekommt,[3] indem er an der ersten Stelle der Relation notiert wird und somit jedem Kunststoff der entsprechende Attributwert zugeordnet wird. Da der `identifier` hier angegeben werden muß, funktioniert diese Überführung nur für Individual-Klassen. Zur Darstellung der Attribute von Prototyp-Klassen wird die sortierte Hornlogik (siehe 4.2) verwendet.

[2] In Abschnitt 4.4 werden die implementierten Teile der Transformation als Termersetzungsschemata angegeben. Diese können zur Verdeutlichung der Beispiele herangezogen werden.

[3] Es ist eine Schwäche von ORF, daß der `identifier` für jedes Individuum zur Identifikation definiert werden muß, diese Verwendung aber nicht Teil der Sprachdefinition ist. Der `identifier` wird hier wie ein Primärschlüssel in Datenbanken verwendet.

Die zweite Stelle der Relation enthält den Attributwert; hier sind natürlich alle RELFUN-Konstrukte verwendbar, so daß alle in ORF definierten Werte direkt überführt werden. Es werden allerdings nur die definierten Werte überführt, wodurch in dieser Darstellung nicht das gleiche wie in der ORF-Darstellung ausgedrückt wird, da in ORF alle deklarierten und nicht definierten Attribute mit Variablen belegt werden. Die Abfrage eines solchen Attributs wird nun `unknown` liefern, im Gegensatz zu ORF, wo eine Variable zurückgegeben wird. Es sei an dieser Stelle aber daran erinnert, daß diese Variablen auch in ORF unerwünscht waren, da es sich auch dort eigentlich um 'null values' handelt. Somit wird durch diese Transformation ein Problem der gegenwärtigen ORF-Version gelöst.

Dadurch können Variablen nun eindeutig für Koreferenzen verwendet werden, wobei die Überführung aber nicht ganz einfach ist, da die Koreferenz zwischen zwei Attributen nicht mehr durch die Verwendung von gleichen Variablenbezeichnern innerhalb einer einzigen Definition ausgedrückt werden kann. Aber auch Koreferenzen können auf die folgende Weise abgebildet werden:

ORF:

```
novodur(
  identifier[novodur_r_5320],
  density[X],
  tension_module_of_elasticity[X]]).
```

Attributzentriert:

```
density(novodur_r_5320, X).
tension_module_of_elasticity(novodur_r_5320, X) :-
  density(novodur_r_5320, X).
```

Die automatische Überführung dieser Koreferenzen ist allerdings sehr aufwendig, da ein automatischer Übersetzer sie erst entdecken muß. Daher ist ihre Transformation nicht in den erstellten Übersetzer eingebaut. Dies ist bei der Transformation von RTPLAST kein Problem, da Koreferenzvariablen in der derzeitigen RTPLAST-Version nicht vorkommen.
Die Vorteile dieser attributzentrierten Darstellung sind folgende:

- Sie ist vollständig deklarativ.
- Auf dieser Version gewonnene Ergebnisse der Evolution können direkt, d. h. ohne Transformation, in die KB zurückgespeist werden.

- Sie ist leicht erweiterbar; d. h. bei zusätzlichen Einträgen müssen die bestehenden Relationen und Regeln nicht verändert werden.
- Die zu einer Materialeigenschaft (Attribut) gehörenden Objekte sind nicht mehr über die gesamte KB verstreut, wodurch Attribut-Updates erschwert waren, was allerdings auch durch eine *attributzentrierte* Oberflächenpräsentation hätte gelöst werden können.
- Sie enthält keine die Evolution störenden Deklarationen.
- Die Attributnamen bleiben im Gegensatz zur n-stelligen Darstellung bei den Wissenseinträgen erhalten.

Die Darstellung erfüllt also schon einige der wichtigsten Forderungen, sie hat allerdings auch Nachteile:

- Die zu einem Kunststoff (Objekt) gehörenden Attribute können über die ganze KB verstreut sein, wodurch Objekt-Updates erschwert werden können. Dieses Problem ließe sich allerdings durch eine *objektzentrierte* Oberflächenpräsentation oder ein Modularisierungskonzept lösen, das durchaus deklarativ zu realisieren ist (siehe Teiltheorien aus ML^2 [vHB92]).
- Durch die Angabe des Identifiers für jedes Attribut wird viel überflüssiger Speicherplatz verbraucht, und alle Evolutions-Algorithmen müssen die zu einem Kunststoff gehörenden Attribute über diesen Identifier aufsammeln.
- Die parallele multiple Vererbung (siehe Abschnitt 4.2.2) kann im Vergleich zur positionalen Darstellung zu Effizienzverlusten führen. Dies ist jedoch nur ein Problem bei tiefen Hierarchien.

Diese Nachteile werden mit Einschränkungen jedoch durch die Vorteile aufgewogen.

4.2 Sortierte Hornlogik und Klassen

Um eine wenig redundante Speicherung der Fakten zu erreichen, wurde in ORF mit Hilfe der Prototyp-Klassen eine Vererbungshierarchie aufgebaut, mit der Attributwerte an die tiefer liegenden Klassen vererbt wurden. Für die Übertragung dieser Vererbung in die attributzentrierte Darstellung gibt es zwei Möglichkeiten:

1. Die Werte werden während der Überführung statisch an die Individual-Klassen vererbt, d. h. für jedes Individuum, das in der Vererbungskette liegt, werden die vererbten Attribute als Relation erzeugt. Diese Darstellung hat zwar den Vorteil das sie für Anfragen zeiteffizient zu verarbeiten ist, allerdings speichert sie viele redundante Fakten, wodurch nicht nur ein größerer Bedarf an Speicherplatz ensteht, sondern auch die Pflege der KB erschwert wird.

2. Eine Aufgabe der Exploration ist es ja gerade, solche Vererbungsmöglichkeiten zu entdecken, weshalb es eine Möglichkeit zur Darstellung einer (dynamischen) Vererbung geben muß, und bereits bestehende Vererbungsbeziehungen bei der Überführung nicht verloren gehen sollten. Deshalb wird in der attributzentrierten Darstellung die Vererbung über Sorten realisiert, die als Erweiterung der VEGA-Sprache zugelassen sind.

4.2.1 Darstellung der Prototyp-Klassen

Zur Überführung der Definitionen von Prototyp-Klassen wird wie für die Individual-Klassen aus jedem definierten Attribut eine zweistellige Relation erzeugt; allerdings wird an der ersten Stelle der Relation nicht mehr der Identifier notiert, sondern eine Variable angegeben, die von eben der Sorte ist, die vorher die Prototyp-Klasse bezeichnet hat. Für die bekannte Definition sieht das wie folgt aus:

ORF:

```
absc(
 ball_thrust_hardness[90],
 additives[[]]).
```

Sortiert:

```
ball_thrust_hardness(X:absc, 90).

additives(X:absc, []).
```

Bei der hier für Präsentationszwecke gewählten Darstellung wird durch den Doppelpunkt gekennzeichnet, daß die Variable `X` von der Sorte `absc` ist. Die Bedeutung der Sorte ist hierbei folgende: Alle Elemente der Sorte `absc` und ihrer Untersorten können an die Variable `X` gebunden werden, wodurch die hierdurch repräsentierte Klasse für eben diese Elemente definiert ist, d. h. unifiziert werden kann.

Es existieren zwar Algorithmen, die diese im Kopf notierten Sorten während der Unfikation verarbeiten können, doch wurde die RELFUN-Unifikation noch nicht um diese Verarbeitung erweitert,[4] so daß eine Transformation der Sorten in unäre Prädikate nötig ist, die jedoch bereits existiert und intern verwendet wird; Validierer und

[4]Diese Erweiterung hat gewöhnlich zur Folge, daß die Sorte in der Darstellung kein normales Prädikat ist, sondern durch den : typartig ausgezeichnet wird; in [Bol94] wird angedeutet, wie unäre Prädikate *als* Sorten verwendet werden können. Zum anderen kann durch die Verarbeitung der Sorte im Klauselkopf eine erhebliche Effizienzsteigerung erzielt werden [BBDV91].

Explorierer aber weiterhin auf der sortierten Notation arbeiten. Da an der transformierten Darstellung die Vererbung leichter zu erklären ist, wird sie an dieser Stelle auch gezeigt:

Unäre Prädikate:

```
ball_thrust_hardness(X, 90) :-
    absc(X).

additives(X, []) :-
    absc(X).
```

An diesem Beispiel ist zu erkennen, wie die Sorte als Prädikat im Klauselrumpf verwendet wird, wodurch die Definition für alle Elemente der Sorte `absc` gilt. Wie die Definition auf alle Untersorten der Sortenhierarchie überführt wird, erklärt der nächste Abschnitt.

4.2.2 Darstellung der Sortenhierarchie

Die Darstellung der einzelnen Elemente einer Sorte wird durch ein Prädikat realisiert. Aus jeder Definition einer Individual-Klasse wird ein solches Prädikat erzeugt, dessen Name dem Klassennamen entspricht und dessen Wert der Identifier ist. Das folgende Beispiel soll dies verdeutlichen:

ORF:

```
novodur(
  identifier[novodur_r_5320],
  density[0.84],
  tension_module_of_elasticity[2000]).
```

Extrahiertes Sortenprädikat:

```
novodur(novodur_r_5320).
```

Mit dieser Definition ist die Vererbung allerdings noch nicht realisiert, da erst noch die Vererbungsbeziehung dargestellt werden muß. Hierfür werden die in ORF durch `super[...]` deklarierten Beziehungen in eine Klausel umgeformt, deren Kopf und Rumpf aus jeweils einem Prädikat bestehen, wodurch die Vererbungsbeziehung auf folgende Weise ausgedrückt wird:

ORF-Deklaration:

```
declare(indi-class[novodur,
                   super[absc], ...).
```

Sorten-Regel:

```
absc(X) :-
    novodur(X).
```

Diese Regel drückt aus, daß jedes `novodur` auch ein `absc` ist und daß alle für `absc` definierten Attribute auch für `novodur` gelten. Dadurch kann die Vererbung direkt in RELFUN auf die folgende Weise abgearbeitet werden:

Gegeben sei folgendes Top-Level-Goal:

```
rfi-p> ball_thrust_hardness(novodur_r_5320, Ball)
```

Auf das Goal kann dann die nächste Regel angewendet werden, welche `X` an `novodur_r_5320` und `Ball` an `90` bindet:

```
ball_thrust_hardness(X, 90) :-
    absc(X).
```

Auf `absc(X)` wird dann mit `X=novodur_r_5320` die folgende Regel angewendet:

```
absc(X) :-
    novodur(X).
```

Durch den untenstehenden Fakt führt der Goal zum Erfolg:

```
novodur(novodur_r_5320).
```

Das Goal ist also erfolgreich und liefert die Bindung der Variable `Ball` an `90`. Das zeigt wie die Vererbung deklarativ dargestellt werden kann und somit auch in jedem Prolog-System zu verarbeiten ist, wodurch eine hohe Wiederverwendbarkeit von RTPLAST erreicht wird.

Die mit Hilfe der Sorten realisierte Vererbung entspricht nicht genau der von ORF. Während Mehrfachdefinitionen von Attributwerten innerhalb einer ORF-Vererbungskette unifiziert werden, kann ein Goal diese in der sortierten Darstellung nachfordern. Es sei z. B. zusätzlich das folgende Sortenprädikat definiert:

```
ball_thrust_hardness(X, 82) :-
    absc(X).
```

Durch Nachfordern würde nun `Ball` an `90` und `82` gebunden. Da RELFUN in diesem Fall zuerst den als Fakt definierten Wert des Individuums liefert, ist das Überschreiben von vererbten Werten in der sortierten Darstellung möglich. Ob dies ein Vor- oder Nachteil ist, kann an dieser Stelle nicht entschieden werden. Eine genaue Erörterung dieser Frage geht über den Rahmen dieser Arbeit hinaus. Die Entdeckung von inkonsistent mehrfach definierten Werten kann als Aufgabe des Validierers betrachtet werden.

4.2.3 Weitergehende Verwendung der Sorten

Natürlich kann mit Sorten mehr erreicht werden, als die effizientere Abarbeitung und Darstellung von Vererbungsbeziehungen. So kann z. B. ein Teil der Funktionalität von KL-ONE-ähnlichen Sprachen realisiert werden, nämlich diejenigen Konstrukte, die sich hornlogisch darstellen lassen. Der $\sqcap$-Operator (siehe [BS85]) kann z. B. durch Prämissen-Konjunktion nachgebildet werden:

Konceptdefinition:

`novodur` $=$ `absc` $\sqcap$ $\exists$ `producer.bayer`

Logisch (Prolognotation):

```
novodur(X):-
    absc(X),
    producer(X, Y:bayer).
```

Dadurch ist es möglich, einige hilfreiche Dienste terminologischer Systeme auf der Basis des Classifiers [ADH91] auch für die Evolution zu nutzen. Die vorsichtige Erweiterung der Sorten ist ein Thema von VEGA.

4.3 Überführen der Anfragen

In den folgenden zwei Unterabschnitten wird erläutert, wie ORF-Anfragen in die attributzentrierte Darstellung überführt werden. Hierbei sind Anfragen, die direkt an Klassen gerichtet sind, zu unterscheiden von solchen, die Klassen hierarchisch durchsuchen. Die beschriebenen Überführungen können hierbei auf drei verschiedene Arten verwendet werden:

1. Sie übersetzen in der ORF-KB selbst enthaltene Anfragen, wie sie z. B. in den Regeln benutzt werden. Hierfür wird jedes Auftreten von Anfragen in Klauseln durch die unten beschriebenen Übersetzungen substituiert. Dies wurde nur für den `instance>`-Operator implementiert, da nur er in RTPLAST verwendet wird.

2. Sie können in den RELFUN-Emulator integriert werden, so daß der Benutzer weiterhin die ORF-Konstrukte für Anfragen benutzt, intern aber die attributzentrierte Darstellung verwendet wird. Dies wurde jedoch ebenfalls nur für den `instance>`-Operator implementiert.

3. Sie können dem Benutzer als Anleitung zur Erstellung von Anfragen an die attributzentrierte Darstellung dienen, falls dieser an die ORF-Anfragen gewöhnt ist, jetzt aber direkt auf einer attributzentrierten KB arbeitet.

Welche Verwendung benutzt wird, hängt von der Entscheidung ab, ob die KB weiterhin in der ORF-Version oder direkt in der attributzentrierten Darstellung gepflegt werden soll.

4.3.1 Klassen-Anfragen

In diesem Abschnitt wird beschrieben, wie die Anfragen an ORF-Klassen in die attributzentrierte Darstellung überführt werden. Die Überführung wird wie folgt realisiert: Jedes in der Anfrage gewünschte Attribut wird in eine zweistellige Relation umgewandelt, die dann zu einer Konjunktion zusammengefaßt werden, wobei die Konjunkte wie üblich durch ein Komma getrennt sind. Die zweite Stelle jeder Relation wird dabei mit dem in der Anfrage gegebenen Attributwert besetzt, wobei es egal ist, ob hier eine Konstante oder Variable definiert wurde.

Eine Ausnahme bildet der `identifier`. Hat er einen konstanten Wert, werden also Attribute eines konkreten Kunststoffs erfragt, so werden mit dem Wert des `identifiers` die ersten Stellen der Relationen besetzt. Ist der Wert des `identifier` eine Variable, so werden die ersten Stellen mit eben dieser Variablen besetzt, wobei sie von der Sorte ist, die der Klasse der Anfrage entspricht. Ist der `identifier` nicht angegeben, so wird anstatt einer bestimmten Variablen eine beliebige, vorher nicht verwendete, Variable an die ersten Stellen der Relationen gesetzt. Die Sorte dieser Variablen muß ebenfalls der Klasse aus der Anfrage entsprechen. Für alle zu erzeugenden Relationen ist hierbei die gleiche Variable zu verwenden.

ORF-Anfrage mit konstantem `identifier`:

```
novodur(
  identifier[novodur_r_5320],
  density[Density],
  tension_module_of_elasticity[E-module])
```

Attributzentrierte Anfrage:

```
density(novodur_r_5320, Density),
              tension_module_of_elasticity(novodur_r_5320, E-module)
```

ORF-Anfrage mit variablem `identifier`:

```
novodur(
  identifier[Ident],
  density[0.84],
  tension_module_of_elasticity[E-module])
```

Attributzentrierte Anfrage:

```
density(Ident:novodur, 0.84),
               tension_module_of_elasticity(Ident:novodur, E-module)
```

Nachdem diese Umformung vorgenommen wurde, kann die gesamte Klassen-Anfrage durch die Konjunktion ersetzt werden, welche nun die gleiche Funktionalität wie die Anfrage besitzt.

4.3.2 `instance>`-Operator

Da in RTPLAST nur der `instance>`-Operator benutzt wird, soll von den hierarchischen Anfrage-Operatoren auch nur dessen Überführung in diesem Abschnitt beschrieben werden, die der Übersetzung der Klassen-Anfragen sehr ähnlich ist.

Der `instance>`-Operator wird wie die Klassen-Anfrage in eine Konjunktion der zweistelligen Relationen überführt. Alle Attribute, einschließlich des `identifiers` mit seiner Sonderstellung, werden ebenfalls wie bei einer Klassen-Anfrage übersetzt. Der einzige Unterschied zur Klassen-Anfrage ist, die an den ersten Stellen der Relationen zu definierende Sorte, falls dort eine Variable zu notieren ist. Die hier zu verwendende Sorte muß der Klasse entsprechen, ab der die Suche beginnt, also der Klasse die an der ersten Stelle des `instance>`-Operators steht. Der einzige Fall für den keine Variable an der ersten Stelle notiert wird, ist die Definition des `identifiers` mit einer Konstanten in der Anfrage, wofür die erste Stelle der Relationen mit dem Wert des `identifiers` besetzt wird.

Der so erstellten Konjunktion muß ein Konjunkt zugefügt werden, mit dem die für die gefundene Klasse zu bindende Variable ermittelt wird (`Class`). Hierfür

wird ein higher-order Prädikat in die Konjunktion eingefügt, dessen Bezeichner eben die gesuchte Variable ist (`Class`), und das auf die an der ersten Stelle der Relationen notierte Variable oder Konstante angewendet wird, wodurch die Unifikation die Sortendefinition des Kunststoffs auf dieses Prädikat anwendet (z. B. `novodur(novodur_r_5320)`). Dieses Vorgehen soll an den folgenden Beispielen verdeutlicht werden:

`instance>`-Anfrage mit konstant definiertem `identifier`:

```
instance>(thermoplastic(),
          Class(identifier[novodur_r_5320],
                tension_module_of_elasticity[E-module],
                density[Density]))
```

Konjunktion als Anfrage:

```
tension_module_of_elasticity(novodur_r_5320, E-module),
                                    density(novodur_r_5320, Density),
                                                Class(novodur_r_5320)
```

`instance>`-Anfrage mit variabel definiertem `identifier`:

```
instance>(thermoplastic(),
          Class(identifier[Ident],
                tension_module_of_elasticity[E-module],
                density[0.84]))
```

Konjunktion als Anfrage:

```
tension_module_of_elasticity(Ident:thermoplastic, E-module),
                                  density(Ident:thermoplastic, 0.84),
                                                      Class(Ident)
```

Mit dieser Umformung ist die Beschreibung der Überführung der ORF-Version von RTPLAST in die attributzentrierte Version vollständig. Ein automatischer Übersetzer für die in RTPLAST verwendeten ORF-Konstrukte wurde implementiert und auf RTPLAST angewendet. Die mit dieser Transformation erzeugte Repräsentation ist in Anhang B abgedruckt. Da die attributzentrierte Darstellung den für die evolutionsgerechte Sprache gestellten Anforderungen gerecht wird, kann sie als Kern dieser Sprache benutzt werden. Das Ziel der Überführung der in einer vorläufigen Sprache erstellten KB in die VEGA-Sprache ist somit erreicht.

4.4 Termersetzungsschemata der implementierten Transformationen

Im folgenden Abschnitt werden die im Rahmen dieser Arbeit implementierten Transformationen als Termersetzungsschemata beschrieben. Da in RTPLAST nur die Prototyp-Klassen, die Individual-Klassen und der `instance>`-Operator verwendet werden, wurde auch nur deren Transformation implementiert.

4.4.1 Termersetzungsschema der Individual-Klassen

indi-class`(identifier[`*name*`],`
attr1`[`*val1*`],`
...
attrN`[`*valN*`]).`
$\Longrightarrow$
indi-class(*name*).
attr1(*name*, *val1*).
...
attrN(*name*, *valN*).

Synbole werden hierbei kursiv und Konstanten in Maschinenschrift notiert.

4.4.2 Termersetzungsschema der Prototyp-Klassen

proto-class`(`*attr1*`[`*val1*`],`
...
attrN`[`*valN*`]).`
$\Longrightarrow$
attr1(*X*:*proto-class*, *val1*).
...
attrN(*X*:*proto-class*, *valN*).

Bei der Transformation sei X eine Variable, für die $X \not\equiv$ *val1*,...,*valN* gelte.

4.4.3 Termersetzungsschema des `instance>`-Operators

Mit angegebenem `identifier`:

`instance>(`*upper-class*`(),`
lower-class`(identifier[`*id-name*`],`
attr1`[`*val1*`],`
...
attrN`[`*valN*`]))`
$\Longrightarrow$
attr1(*id-name*: *upper-class*, *val1*),
...
attrN(*id-name*: *upper-class*, *valN*),
lower-class(*id-name*)

Ohne angegebenem `identifier`:

`instance>(`*upper-class*`(),`
lower-class`(`*attr1*`[`*val1*`],`
...
attrN`[`*valN*`]))`
$\Longrightarrow$
attr1(X: *upper-class*, *val1*),
...
attrN(X: *upper-class*, *valN*),
lower-class(*id-name*)

Es gelte wiederum $X \not\equiv$ *val1*,...,*valN*.

Kapitel 5

Zusammenfassung und Ausblick

Aufgabe dieser Diplomarbeit war die prototypische Erstellung einer KB, die als Kern einer bereits konzipierten KB zur recyclinggerechten Produkt- und Produktionsplanung einsetzbar ist [BBK93, Kre94]. Aus der Fülle der Module dieser KB wurden die Materialien als Kern ausgewählt, da sie zum einen die Grundlage jeder technischen Produktentwicklung und zum anderen bereits ohne weitere Module der RPPP-KB zu verwenden sind, wodurch sichergestellt ist, daß bereits der erste KB-Prototyp Anwenderfeedback erlaubt.

Für die Prototypisierung des in diesem Gebiet erhobenen Wissens wurde eine objektzentrierte Darstellung verwendet, die dem Benutzer eine leichte Erweiterung und Anwendung der KB ermöglicht. Diese objektzentrierte Darstellung kann in einer logischen Programmiersprache verarbeitet werden, da sie durch eine Vorverarbeitung in eine positionalisierte relationale Form überführt wird. Da die positionalisierte Form vollständig deklarativ ist, kann diese als Testobjekt für die Evolutions-Algorithmen verwendet werden. Hierbei tritt allerdings das Problem auf, daß durch Validierung und vor allem durch Exploration gewonnene Ergebnisse nicht einfach in die objektzentrierte Form zurückgeführt werden können.

Nachdem sich herausgestellt hatte, daß die Verwendung der zwei verschiedenen Darstellungen die oben beschriebenen Probleme verursacht, wurde die attributzentrierte Darstellung entworfen, die den Vorteil hat, daß sie zum einen deklarativer und zum anderen ebenso flexibel wie die objektzentrierte Darstellung ist. Hierdurch ist es möglich, RTPLAST in Zukunft in einer einzigen Version zu halten. Zur Überführung der objektzentrierten in die attributzentrierte Darstellung wurde ein Übersetzer implementiert, der Teil des RELFUN-Systems ist.

Mit dieser Arbeit wurde ein Software-Grundstein für die im Themengebiet VEGA zu lösenden Probleme gelegt. Die weiteren aus dieser Arbeit resultierenden Aufgaben sind folgende: RTPLAST sollte auf eine für die Praxis realistische Größe erweitert werden. Dies kann zum Teil mit einer automatischen Übertragung der Fakten aus bestehenden Datenbanken realisiert werden. Weiterhin ist zu prüfen, wie die deklarative Darstellung in anderen Wissensgebieten zu verwenden ist und welche Erweiterungen der sortierten hornlogischen Sprache hierfür nötig sind. So begann aufbauend auf

dieser Diplomarbeit eine Projektarbeit am DFKI [Pos94], in der unter anderem auch eine Teil-KB über Fertigungsprozesse in derselben Sprache dargestellt werden soll.

Die Einführung der Sorten wirft ebenfalls einige Fragen auf: So ist zum einen zu klären, welche Sortenverwendungen weiterhin statische Verarbeitung nutzen sollten, d. h. eine Vorverarbeitung, und welche die um Sorten zu erweiternde dynamische Unifikation [Bol94]. Dieses Thema wird in einer am DFKI geplanten Projektarbeit behandelt. Zum anderen sollte untersucht werden, welche Erweiterungen von Sorten in Richtung KL-ONE-artiger Sprachen sinnvoll sind [Han93], also welche Ausdrucksmächtigkeit mit welchem Aufwand gewonnen werden kann. Weiterhin sollte geklärt werden, inwieweit die Sequentialisierung der parallelen multiplen Vererbung (im Sortenverband) zu Effizienzproblemen führt.

Schließlich sind die noch fehlenden Erweiterungen der sortierten hornlogischen Kernsprache in Richtung auf ein hybrides System mit dem Umfang des vollen COLAB [BHHM91] eine offene Frage, die nur im Zusammenhang mit weiteren konkreten Anwendungen und Evolutionsnotwendigkeiten gelöst werden kann.

Literaturverzeichnis

[ADH91] A. Abecker, D. Drollling, P. Hanschke. TAXON: A Concept Language with Concrete Domains. In Michael M. Richter, Harold Boley (Hrsg.), *Preprints of the Proceedings of the International Workshop on Processing Declarative Knowledge (PDK'91)*. DFKI (German Research Center for Artificial Intelligence), June 1991.

[AKN86] Hassan Aït-Kaci, Roger Nasr. Login: a logic programming language with built-in inheritance. *The Journal of logic programming*, 3:185–215, 1986.

[AL88] Hassan Aït-Kaci, Patrick Lincoln. LIFE: A Natural Language for Natural Language. MCC Technical Report ACA-ST-074-88, MCC, ACA Program, February 1988.

[Bar91] A. Barg. Recyclinggerechte Produkt- und Produktionsplanung. *VDI-Z*, 11, 1991.

[BBDV91] A. Bockmayer, C. Brzoska, P. Deussen, I. Varsek. KA-Prolog: Erweiterungen einer logischen Programmiersprache und ihre effiziente Implementierung. *Informatik Forschung und Entwicklung*, 6:128–140, 1991.

[BBG93] H. Boley, F. Bry, U. Geske (Hrsg.). *Proc. Workshop "Neuere Entwicklungen der deklarativen KI-Programmierung" auf der KI-93, Humboldt-Univ. zu Berlin*, Research Report RR-93-35, September 1993. DFKI Kaiserslautern.

[BBK93] Harold Boley, Ulrich Buhrmann, Christof Kremer. Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien. DFKI Technical Memo TM-93-03, DFKI, August 1993. Presented at Workshop 'KI und Umweltanwendungen', KI-93, Berlin.

[BDS90] H. Breuer, G. Dupp, J. Schmitz. Einheitliche Werkstoffdatenbank - eine Idee setzt sich durch. *Kunststoffe 80*, 11, 1990.

[BEH+93] Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein. RELFUN Guide: Programming with Relations and Functions Made Easy. Document D-93-12, DFKI, July 1993.

[BHH+92] Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer, Michael M. Richter. *VEGA – Knowledge Validation and Exploration by Global Analysis*. Project Proposal, DFKI Kaiserslautern, October 1992.

[BHHM] H. Boley, P. Hanschke, K. Hinkelmann, M. Meyer. COLAB: A Hybrid Knowledge Representation and Compilation Laboratory. To appear in: *Annals of Operations Research*. Presented at 3rd International Workshop on Data, Expert Knowledge and Decisions: Using Knowledge to Transform Data into Information for Decision Support, Reisensburg, Germany.

[BHHM91] Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer. COLAB: A Hybrid Knowledge Compilation Laboratory. Presented at 3rd International Workshop on Data, Expert Knowledge and Decisions: Using Knowledge to Transform Data into Information for Decision Support, Reisensburg, Germany, September 1991.

[BMPS+90] Ronald J. Brachmann, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, Alexander Borgida. Living with CLASSIC: When and How to Use a KL-ONE -Like Language. In *Principles of Semantic Networks*. J. Sowa Morgan Kaufmann Publishers Inc, Juni 1990.

[Bol90] Harold Boley. Expert system shells: very-high-level languages for Artificial Intelligence. *Expert Systems*, 7(1):2–8, February 1990.

[Bol91] Harold Boley. A Sampler of Relational/Functional Definitions. DFKI Technical Memo TM-91-04, DFKI, March 1991. Second, Revised Edition July 1993.

[Bol94] Harold Boley. Finite Domains and Exclusions as First-Class Citizens. In Roy Dyckhoff (Hrsg.), *Fourth International Workshop on Extensions of Logic Programming, St. Andrews, Scotland, 1993, Preprints and Proceedings*, LNAI. Springer, March 1994.

[BS85] R. J. Brachman, J. G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[CR91] H. Corsten, M. Reiss. Recycling in PPS-Systemen. *Die Betriebswirtschaft*, 51, 1991.

[Cze92] D. Czedik. Status Quo der Wiederverwendbarkeit von Wissensbasen. *KI*, 1:27–39, 1992.

[Eve86] Gordon C. Everest. *Database Managment: Objectives, System Functions, and Administration*. McGraw-Hill, 1986.

[Han93] Philipp Hanschke. A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning. Research Report RR-93-46, DFKI, 1993.

[HFRF90] I. Hulthage, M. S. Fox, M. D. Rychener, M. L. Farinacci. The Architecture of Aladin: A Knowledge-Based Approach to Alloy Design. *IEEE Expert*, 5:56–73, 1990.

[HPFR87] I. Hulthage, M. A. Przystupa, M. L. Farinacci, M. D. Rychener. The Representation of Metallurgical Knowledge for Alloy Design. *AI EDAM*, 1(3):159–168, 1987.

[KLW93] J. Kunz, W. Land, J. Wiener. *Neue Konstruktionsmöglichkeiten mit Kunststoffen durch schnelle und sichere Werkstoffauswahl.* WEKA Fachverlag für technische Führungskräfte GmbH, Augsburg, 1993.

[KMK91] Kenneth A. Kaufman, Ryszard S. Machalski, Larry Kerschberg. An Architecture for Integrating Machine Learning and Discovery Programs into a Data Analysis System. In *AAAI ,Knowledge Discovery in Databases Workshop*, 1991.

[Kre94] Christof Kremer. Konzeption einer Deklarativen Wissensbasis über Technisches Recycling. Diplomarbeit, Universität Kaiserslautern, FB Informatik, Postfach 3049, 67608 Kaiserslautern, Januar 1994.

[Kue93] Otto Kuehn. Knowledge Sharing and Knowledge Evolution. In *Proceedings of IJCAI'93 Workshop on Knowledge Sharing and Information Interchange*, 1993.

[LG89] Douglas B. Lenat, R. V. Guha. *Building Large Knowledge-Based Systems.* Addison-Wesley Publishing Company, Inc., Dezember 1989.

[MW93] Gregor Meyer, Sybilla Weigel. Polymorphe Featuretypen - Typinferenz und Typüberprüfung. In Boley et al. [BBG93].

[NFF+91] R. Neches, R. Fikes, F. Finin, T. Gruber, R. Patil, T. Senator, R. Swartout, William. Enabling technology for knowledge sharing. *AI-Magazin*, 12(3):36–53, 1991.

[Nys85] Sven Olof Nystrøm. NyWam - A WAM Emulator Written in LISP. 1985.

[PB93] Heribert Popp, Josef Barthel. Kommunikation im hybriden Expertensystem ELDAR zur wissensbasierten Stoffdatenversorgung. In F. Puppe, A. Günter (Hrsg.), *Expertensysteme 93*, Band 2, S. 222–234. Springer-Verlag, Februar 1993.

[Pos94] S. Possner. Aufbau und Evolution einer Wissensbasis über Verbundwerkstoffe sowie deren Fertigung. Projektarbeit, Universität Kaiserslautern, FB Informatik, Postfach 3049, 67608 Kaiserslautern, März 1994.

[PSK+93] R. Pitchumani, P. A. Schwenk, V. M. Karbhari, J. F. Ramsay, T. D. Claar. A Knowledge-Based Decision Support System for the Manufacture of Composite Preforms. 1993.

[Rah93] Jörg Rahmer. Bewertender Vergleich von Wissensaustauschformaten. Diplomarbeit, Universität Kaiserslautern, FB Informatik, Postfach 3049, D-6750 Kaiserslautern, Oktober 1993. In German.

[RBB+93] M.M. Richter, B. Bachmann, A. Bernardi, C . Klauck, R. Legleitner, G. Schmidt. Von IDA zu IMCOD: Expertensysteme im CIM–Umfeld. Research Report RR–93–36, DFKI, Juli 1993.

[Sin93] Michael Sintek. ORF — Object-Centered RELFUN. DFKI Kaiserslautern, November 1993.

[SPM93] K. H. Simon, B. Page, A. Manche. Expertensystemanwendungen im Umweltschutzbereich: Konzepte, System, Probleme. In *Materialien zum Worshop 5*. XPS-93, 1993.

[SSB91] M. Sintek, W. Stein, U. Buhrmann. Validation and Exploration of the Period System of the Elements: A RELFUN Knowledge Base. In SiSt-Bu93 [Bol91]. Second, Revised Edition July 1993.

[Ull88] Jeffrey D. Ullman. *Database and Knowledge-Base Systems*, Band 1. Computer Science Press, 1988.

[vdVM91] Paul E. van der Vet, Nicolaas J. I. Mars. An ontology of ceramics. Memoranda Informatica 91-85, University of Twente, TO/INF library, The Memoranda Informatica Secretary, P.O. Box 217, 7500 AE Enschede, The Netherlands, Dezember 1991.

[Ver93] Verein Deutscher Ingenieure (Hrsg.). *Recyclinggerechte Produktentwicklung. Aspekte, Strategien, Konstruktionspraxis*, Band 1089. Verein Deutscher Ingenieure - Verlag, November 1993.

[vHB92] Frank van Harmelen, John Balder. $(ML)^2$: A formal language for KADS model. In *Models for Problem solving*, 1992.

[War83] David H. D. Warren. An Abstract Prolog Instruction Set. Technical Note 309, SRI International, Menlo Park, CA, October 1983.

[wg93] The VEGA-KBS working group. A Knowledge Base for Recycling-oriented Product and Production Planning: Design Issues, Informational Contents and Formal Representability. VEGA Milestone 93-01, DFKI GmbH, October 1993.

Anhang A

RTPLAST

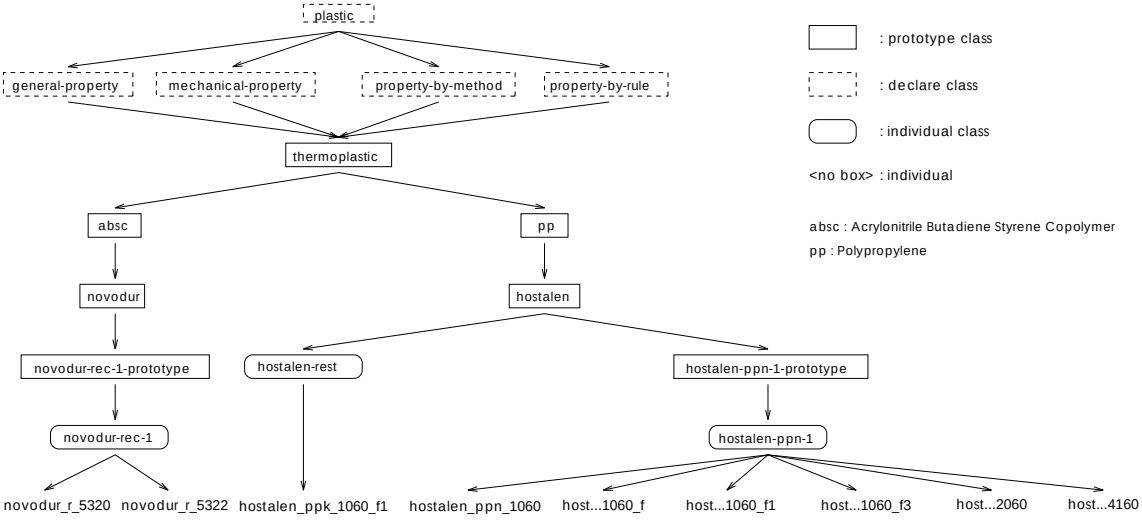

Abbildung A.1: Vererbungshierarchie von RTPLAST

A.1 Die ORF-KB

```
%-------------------------------------------------------------------------------
%|       A Declarative Knowledge Base on Recyclable Thermoplastics             |
%| An object-centeredRelfun Knowledge Base using property terms and inheritance|
%-------------------------------------------------------------------------------

%                                                    December 1993
% (c) Ulrich Buhrmann
%

%-------------------------------------------------------------------------------
% top of all classes:
```

```
declare(proto-class[plastic,
        super[],
        identifier]).
          % all concrete individuals (no prototypes) have an identifier

% proto-classes define classes of prototypes by their
% realizations (plastic(...).)

plastic().

%-------------------------------------------------------------------------------
% the plastics and their properties:

% *-property-* proto-classes are built for structuring of the plastics
% properties, they have
% the meaning of a only declaring class

declare(proto-class[mechanical-property,
        super[plastic],
        yield_stress, yield_elangation, tension_module_of_elasticity,
          tension_module_of_criep_1_h, tension_module_of_criep_1000_h,
          ball_thrust_hardness,
          tension_notched-bar_impact_strength, notched-bar_impact_strength,
          izod-impact_strength_23,
          izod-impact_strength_0, izod-impact_strength_-30,
          izod-notched-bar_impact_strength_23,
          izod-notched-bar_impact_strength_0,
          izod-notched-bar_impact_strength_-30, vicat_a/50,
          vicat_b/50,
          dimensional_stability_hdt/a,
          dimensional_stability_hdt/b]).
  % all attributes names are taken from CAMPUS (of course only if they exist
  % in CAMPUS)

mechanical-property().

declare(proto-class[general-property,
        super[plastic],
        density, melting_index_mfi_1, melting_index_mfi_2,
          melting_index_mfi_3,
          melting_index_mfi_1_temperature, melting_index_mfi_1_stress,
          melting_index_mfi_2_temperature, melting_index_mfi_2_stress,
          melting_index_mfi_3_temperature, melting_index_mfi_3_stress,
          volume_per_unit_time_mvr_1,
          volume_per_unit_time_mvr_1_temperature,
          volume_per_unit_time_mvr_1_stress,
          volume_per_unit_time_mvr_2_temperature,
          volume_per_unit_time_mvr_2_stress,
          volume_per_unit_time_mvr_2,
          recycled, producer, additives]).
```

```
general-property().

declare(proto-class[property-by-method,
        super[plastic],
        stress-strain]).

property-by-method().

declare(proto-class[property-by-rule,
        super[plastic],
        recyclable, used-for]).

property-by-rule().

declare(proto-class[thermoplastic,
        super[general-property, mechanical-property, property-by-method,
          property-by-rule]]).

thermoplastic().

declare(proto-class[pp,
        super[thermoplastic]]).

pp().

declare(proto-class[absc,
        super[thermoplastic]]).

absc().

%-------------------------------------------------------------------------------
% hostalen:

declare(proto-class[hostalen,
        super[pp]]).

% Here you can see how a prototype is defined, all sub-classes of hostalen
% get the values which are defined below.

hostalen(
  producer[hoechst],
  volume_per_unit_time_mvr_1_temperature[190],
  volume_per_unit_time_mvr_1_stress[5],
  volume_per_unit_time_mvr_2_temperature[230],
  volume_per_unit_time_mvr_2_stress[2.16],
  melting_index_mfi_1_temperature[190],
  melting_index_mfi_1_stress[5],
  melting_index_mfi_2_temperature[230],
  melting_index_mfi_2_stress[2.16],
  melting_index_mfi_3_temperature[230],
  melting_index_mfi_3_stress[5],
```

```
  recyclable[recyclable-rule]).

declare(indi-class[hostalen-rest,
        super[hostalen]]).

% indi-classes define individuals of the domain by their
% realization (hostalen-rest(...))

hostalen-rest(
    % hostalen_ppk_1060_f1 is an individual of the plastics domain
  identifier[hostalen_ppk_1060_f1],
  density[0.902],
  melting_index_mfi_1[2],
  melting_index_mfi_2[0.9],
  melting_index_mfi_3[4],
  volume_per_unit_time_mvr_1[2.3],
  volume_per_unit_time_mvr_2[1.3],
    % mechanical properties:
  yield_stress[32],
  yield_elangation[11],
  tension_module_of_elasticity[1300],
  tension_module_of_criep_1_h[850],
  tension_module_of_criep_1000_h[500],
  ball_thrust_hardness[67],
  tension_notched-bar_impact_strength[48],
  notched-bar_impact_strength[8],
  izod-impact_strength_23[100],
  izod-impact_strength_-30[14],
  izod-notched-bar_impact_strength_23[3.5],
  izod-notched-bar_impact_strength_-30[1.5],
  vicat_a/50[152],
  vicat_b/50[88],
  dimensional_stability_hdt/a[56],
  dimensional_stability_hdt/b[94],
  used-for[used-for-rule],
  % only for tests:
  additives[tup[flamability-trigger]]).

declare(proto-class[hostalen-ppn-1-prototype,
        super[hostalen]]).
  % hostalen-ppn-1-prototype is an prototype from which some
  % individuals inherit attribute VALUES

hostalen-ppn-1-prototype(
  density[0.903],
  melting_index_mfi_1[4],
  melting_index_mfi_2[2],
  melting_index_mfi_3[9],
  volume_per_unit_time_mvr_1[4.5],
  volume_per_unit_time_mvr_2[2.7],
    % mechanical properties:
  yield_stress[32],
```

```
  yield_elangation[11],
  tension_module_of_elasticity[1300],
  tension_module_of_criep_1_h[900],
  tension_module_of_criep_1000_h[480],
  ball_thrust_hardness[68],
  tension_notched-bar_impact_strength[45],
  notched-bar_impact_strength[7],
  izod-impact_strength_23[90],
  izod-impact_strength_0[35],
  izod-impact_strength_-30[12],
  izod-notched-bar_impact_strength_23[3.1],
  izod-notched-bar_impact_strength_0[1.6],
  izod-notched-bar_impact_strength_-30[1.4],
  vicat_a/50[152],
  vicat_b/50[89],
  dimensional_stability_hdt/a[56],
  dimensional_stability_hdt/b[90],
  additives[[]]).

declare(indi-class[hostalen-ppn-1,
        super[hostalen-ppn-1-prototype]]).
  % all individuals of hostalen-ppn-1 get the attr-values of
  % hostalen-ppn-1-prototype

hostalen-ppn-1(
  identifier[hostalen_ppn_1060],
  stress-strain[[interpol, [[0.71, 13.4],
                    [1.42, 22.2],
                    [2.13, 26.7],
                    [2.84, 29.5],
                    [3.55, 31.6],
                    [4.26, 33.2],
                    [4.97, 34.3],
                    [5.68, 35.2],
                    [6.39, 35.7],
                    [7.1, 36.0]]]]).

hostalen-ppn-1(
  identifier[hostalen_ppn_1060_f]).

hostalen-ppn-1(
  identifier[hostalen_ppn_1060_f1]).

hostalen-ppn-1(
  identifier[hostalen_ppn_1060_f3]).

hostalen-ppn-1(
  identifier[hostalen_ppn_2060]).

hostalen-ppn-1(
  identifier[hostalen_ppn_4160]).
```

```
%-------------------------------------------------------------------------------
% novodur:

declare(proto-class[novodur,
        super[absc]]).

novodur(
  producer[bayer],
  recyclable[recyclable-rule]).

declare(proto-class[novodur-rec-1-prototype,
        super[novodur]]).

novodur-rec-1-prototype(
  izod-impact_strength_23[60],
  izod-impact_strength_-30[40],
  izod-notched-bar_impact_strength_23[12],
  izod-notched-bar_impact_strength_-30[6],
  ball_thrust_hardness[90],
  recycled[true],
  additives[[]]).

declare(indi-class[novodur-rec-1,
        super[novodur-rec-1-prototype]]).

novodur-rec-1(
  identifier[novodur_r_5320],
  yield_stress[38],
  yield_elangation[2.1],
  tension_module_of_elasticity[2000],
  dimensional_stability_hdt/a[90],
  dimensional_stability_hdt/b[95]).

novodur-rec-1(
  identifier[novodur_r_5322],
  yield_stress[40],
  yield_elangation[2.3],
  tension_module_of_elasticity[2200],
  dimensional_stability_hdt/a[96],
  dimensional_stability_hdt/b[100]).

%-------------------------------------------------------------------------------
% Sometimes attribute values depend on different circumstances, in this       |
% case I represent them as a rule, I use the name of the corresponding rule as |
% attribute value.                                                             |
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
% rule for what kind of product one can use the plastics:

used-for-rule(hostalen_ppk_1060_f1, spritzgiessen) :- &
  [elektrogeraete, technische_teile, haushaltmaschinen, sanitaereinrichtungen,
```

```
   moebelbau, verpackungen].

used-for-rule(hostalen_ppk_1060_f1, pressen) :- &
  [verpackungsbaender].

used-for-rule(hostalen_ppk_1060_f1, blasformen) :- &
  [verpackungsbaender].

%-------------------------------------------------------------------------------
% rule for representing the recyclability:

recyclable-rule(Plastic-id) :-
  instance>(pp(), Class(identifier[Plastic-id], additives[Additives])),
  nonvar(Additives),
  member(flamability-trigger, Additives) &
  only_in_closed_circle.
  % If the pp/abs contains flamability-trigger, one can only recycle the pp/pa
  % in a closed loop, because if the information about the flamability-trigger
  % gets lost in the recycling circle, it is possible that the burning of the
  % plastic at the end of its life-cycle produces dioxin.

recyclable-rule(Plastic-id) :-
  instance>(pp(), Class(identifier[Plastic-id], additives[Additives])),
  nonvar(Additives),
  not-member(flamability-trigger, Additives) &
  possible.
  % If the pp/abs contains no flamability-trigger, then the recycling is
  % possible (not more, not less, just possible)

recyclable-rule(Plastic-id) :-
  instance>(absc(), Class(identifier[Plastic-id], additives[Additives])),
  nonvar(Additives),
  member(flamability-trigger, Additives) &
  only_in_closed_circle.

recyclable-rule(Plastic-id) :-
  instance>(absc(), Class(identifier[Plastic-id], additives[Additives])),
  nonvar(Additives),
  not-member(flamability-trigger, Additives) &
  possible.

% e.g.:
%rfe-p> instance>(thermoplastic(), C(identifier[I],recyclable[R])),
%       nonvar(R),
%       R(I)
%only_in_closed_circle
%C = single-hostalen
%R = recyclable-rule
%I = hostalen_ppk_1060_f1
%rfe-p> m
%possible
%C = hostalen-ppn-1
```

```
%R = recyclable-rule
%I = hostalen_ppn_1060
%rfe-p>

%-------------------------------------------------------------------------------
% This is the part of the KB which represents meta-knowledge.                  |
% If you like to, you can see it as an extended database scheme.               |
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
% attribute classes:

declare(proto-class[attribute,
        super[plastic],
        cardinality, sort]).

attribute().

declare(indi-class[simple-attribute,
        super[attribute]]).

simple-attribute(
  identifier[identifier],
  cardinality[=[1]],
  sort[string]).

simple-attribute(
  identifier[cardinality],
  sort[predicate]).

simple-attribute(
  identifier[sort],
  cardinality[1]).

simple-attribute(
  identifier[method_for_test],
  sort[tupel-of-strings]).

simple-attribute(
  identifier[additives],
  sort[tupel-of-strings]).

simple-attribute(
  identifier[measurement],
  sort[tupel-of-strings]).

simple-attribute(
  identifier[exists-with-conditions],
  sort[tupel-of-strings],
  cardinality[>=[1]]).

simple-attribute(
```

```
  identifier[recyclable],
  sort[boolean]).    % has to be specialized later on

simple-attribute(
  identifier[recycled],
  sort[boolean]).    % has to be specialized later on

%-------------------------------------------------------------------------------
% numerical-attributes:

declare(proto-class[numerical-attribute,
        super[attribute],
        method_for_test, measurement]).
  % proto-class for introducing new attributes (method_for_test, measurement) to
  % subclasses

numerical-attribute(
  cardinality[<=[1]],
  sort[real]).

declare(indi-class[simple-numerical-attribute,
        super[numerical-attribute]]).

simple-numerical-attribute(
  identifier[temperature],
  measurement[c]).

simple-numerical-attribute(
  identifier[density],
  method_for_test[[iso_1183, din_53735]]).

simple-numerical-attribute(
  identifier[volume_per_unit_time_mvr_1_temperature],
  measurement[c]).

simple-numerical-attribute(
  identifier[volume_per_unit_time_mvr_1_stress],
  measurement[kg]).

simple-numerical-attribute(
  identifier[volume_per_unit_time_mvr_2_temperature],
  measurement[c]).

simple-numerical-attribute(
  identifier[volume_per_unit_time_mvr_2_stress],
  measurement[kg]).

simple-numerical-attribute(
  identifier[melting_index_mfi_1_temperature],
  measurement[c]).

simple-numerical-attribute(
```

```
  identifier[melting_index_mfi_1_stress],
  measurement[kg]).

simple-numerical-attribute(
  identifier[melting_index_mfi_2_temperature],
  measurement[c]).

simple-numerical-attribute(
  identifier[melting_index_mfi_2_stress],
  measurement[kg]).

simple-numerical-attribute(
  identifier[melting_index_mfi_3_temperature],
  measurement[c]).

simple-numerical-attribute(
  identifier[melting_index_mfi_3_stress],
  measurement[kg]).

simple-numerical-attribute(
  identifier[vicat_a/50],
  method_for_test[[din_306]],
  measurement[c]).

simple-numerical-attribute(
  identifier[vicat_b/50],
  method_for_test[[din_306]],
  measurement[c]).

simple-numerical-attribute(
  identifier[yield_elangation],
  method_for_test[[iso_527, din_53455]],
  measurement[proz]). % use proz instead of %, because % is reserved for comments

simple-numerical-attribute(
  identifier[dimensional_stability_hdt/a],
  method_for_test[[iso_75, din_53461]],
  measurement[c]).

simple-numerical-attribute(
  identifier[dimensional_stability_hdt/b],
  method_for_test[[iso_75, din_53461]],
  measurement[c]).

%------------------------------------------------------------------------------
% special numerical attributes:

declare(proto-class[numerical-attribute-n/qmm-prototype,
        super[numerical-attribute]]).

numerical-attribute-n/qmm-prototype(
  measurement[/[n, ^[mm, 2]]]).
```

```
declare(indi-class[numerical-attribute-n/qmm,
        super[numerical-attribute-n/qmm-prototype]]).

numerical-attribute-n/qmm(
  identifier[yield_stress],
  method_for_test[[iso_527, din_53455]]).

numerical-attribute-n/qmm(
  identifier[tension_module_of_elasticity],
  method_for_test[[iso_527, din_53457]]).

numerical-attribute-n/qmm(
  identifier[tension_module_of_criep_1_h],
  method_for_test[[iso_899, din_53444]]).

numerical-attribute-n/qmm(
  identifier[tension_module_of_criep_1000_h],
  method_for_test[[iso_899, din_53444]]).

numerical-attribute-n/qmm(
  identifier[ball_thrust_hardness],
  method_for_test[[din_53456]]).

declare(proto-class[numerical-attribute-kj/qm-prototype,
        super[numerical-attribute]]).

numerical-attribute-kj/qm-prototype(
  measurement[/[kj, ^[m, 2]]]).

declare(indi-class[numerical-attribute-kj/qm,
        super[numerical-attribute-kj/qm-prototype]]).

numerical-attribute-kj/qm(
  identifier[tension_notched-bar_impact_strength],
  method_for_test[[iso_8256, din_53448/1b]]).

numerical-attribute-kj/qm(
  identifier[notched-bar_impact_strength],
  method_for_test[[din_53453]]).

numerical-attribute-kj/qm(
  identifier[izod-impact_strength_23],
  method_for_test[[iso_180/1c]]).

numerical-attribute-kj/qm(
  identifier[izod-impact_strength_0],
  method_for_test[[iso_180/1c]]).

numerical-attribute-kj/qm(
  identifier[izod-impact_strength_-30],
  method_for_test[[iso_180/1c]]).
```

```
numerical-attribute-kj/qm(
  identifier[izod-notched-bar_impact_strength_23],
  method_for_test[[iso_180/1a]]).

numerical-attribute-kj/qm(
  identifier[izod-notched-bar_impact_strength_0],
  method_for_test[[iso_180/1a]]).

numerical-attribute-kj/qm(
  identifier[izod-notched-bar_impact_strength_-30],
  method_for_test[[iso_180/1a]]).

declare(proto-class[numerical-attribute-with-conditions,
        super[numerical-attribute],
        exists-with-conditions]).
          % some attributes have to be decribed with the conditions under which
          % they were tested

numerical-attribute-with-conditions().

declare(proto-class[melting_index-prototype,
        super[numerical-attribute-with-conditions]]).

melting_index-prototype(
  sort[real],
  measurement[/[g, *[10, min]]],
  method_for_test[[iso_1133, din_53735]]).

declare(indi-class[melting_index,
        super[melting_index-prototype]]).
  % there are different sorts of melting_index, so it is useful to
  % classify them and save informations about their different properties

melting_index(
  identifier[melting_index_mfi_1],
  exists-with-conditions[[melting_index_mfi_1_temperature,
                          melting_index_mfi_1_stress]]).

melting_index(
  identifier[melting_index_mfi_2],
  exists-with-conditions[[melting_index_mfi_2_temperature,
                          melting_index_mfi_2_stress]]).

melting_index(
  identifier[melting_index_mfi_3],
  exists-with-conditions[[melting_index_mfi_3_temperature,
                          melting_index_mfi_3_stress]]).

declare(proto-class[volume_per_unit_time-prototype,
        super[numerical-attribute-with-conditions]]).
```

```
volume_per_unit_time-prototype(
  sort[real],
  measurement[/[^[cm, 3], *[10, min]]],
  method_for_test[[iso_1133, din_53735]]).

declare(indi-class[volume_per_unit_time,
        super[volume_per_unit_time-prototype]]).

volume_per_unit_time(
  identifier[volume_per_unit_time_mvr_1],
  exists-with-conditions[[volume_per_unit_time_mvr_1_temperature,
                          volume_per_unit_time_mvr_1_stress]]).

volume_per_unit_time(
  identifier[volume_per_unit_time_mvr_2],
  exists-with-conditions[[volume_per_unit_time_mvr_2_temperature,
                          volume_per_unit_time_mvr_2_stress]]).

%-------------------------------------------------------------------------------
% special attributes (which values e.g. are interpolation pairs or rules):

declare(proto-class[interpol-attribute,
        super[attribute],
        first, first-measurement,      % first element is x-axis value
        second, second-measurement]).  % second element is y-axis value
  % form of attribute value:
  % [<method>: interpol, <chart>: [[x-val1, y-val1]], [...], ...]]
  % evaluate with method interpol

interpol-attribute(
  sort[interpol-chart]).

declare(indi-class[simple-interpol-attribute,
        super[interpol-attribute]]).

simple-interpol-attribute(
  identifier[stress-strain],
  first[dehnung],
  first-measurement[proz],
  second[spannung],
  second-measurement[mpa]).

declare(proto-class[rule-attribute,
        super[attribute],
        number-of-parameter]).

rule-attribute(
  sort[rule]).

declare(indi-class[simple-rule-attribute-1,
        super[rule-attribute],
        para]).
```

```
simple-rule-attribute(
  identifier[recyclable],
  number-of-parameter[1],
  para[identifier]).  % identifier of thermoplastic

declare(indi-class[simple-rule-attribute-2,
        super[rule-attribute],
        first-para, second-para]).

simple-rule-attribute(
  identifier[used-for],
  number-of-parameter[2],
  first-para[identifier],   % identifier of thermoplastic
  second-para[processing]). % used-for value depends on the sort of processing

%-------------------------------------------------------------------------------
% In this part one can find methods for, e.g., calculating something from    |
% the attribute values or methods which are used in some place earlier in    |
% the KB.                                                                     |
%-------------------------------------------------------------------------------

%-------------------------------------------------------------------------------
% extra methods:

% this is a method for interpolation
% for prototyping reasons this is the implementation of the Lagrange algorithm:

interpol(Pairs, X) :- &     % Pairs is the value of the attribute and
  sum(Pairs, X, Pairs, 1).  % X is the value which has to be calculatet

sum(Pairs, X, [], No) :- & 0.

sum(Pairs, X, Left-over-pairs, No) :-
  [[First, Second] |Left-over] is Left-over-pairs &
  +(*(Second, mul(Pairs, X, First, No, 1)), sum(Pairs, X, Left-over, +(No, 1))).

mul([], X, Xv, No, Mulno) :- & 1.

mul(Pairs, X, Xv, No, No) :-
  [_ |Left-over] is Pairs &
  mul(Left-over, X, Xv, No, +(No, 1)).

mul(Pairs, X, Xv, No, Mulno) :-
  [[First, _] |Left-over] is Pairs,
  /=(0, -(Xv, First)),
  Result is /(-(X, First), -(Xv, First)) &
  *(Result, mul(Left-over, X, Xv, No, +(Mulno, 1))).

%e.g.:
%rfe-p> instance>(thermoplastic(), C(identifier[I],
%                                    stress-strain[[Method, Chart]])),
```

```
%        nonvar(Method),
%        Method(Chart,2.3)
%27.473159400564154
%%C = hostalen-ppn-1
%I = hostalen_ppn_1060
%Method = interpol
%Chart = [[0.71, 13.4],
%         [1.42, 22.2],
%         [2.13, 26.7],
%         [2.84, 29.5],
%         [3.55, 31.6],
%         [4.26, 33.2],
%         [4.97, 34.3],
%         [5.68, 35.2],
%         [6.39, 35.7],
%         [7.1, 36.0]]

%------------------------------------------------------------------------------
% auxiliary mehtods/predicates:

not-member(Element, []).

not-member(Element, [First | Left-over]) :-
  string/=(princ-to-string(First), princ-to-string(Element)),
  not-member(Element, Left-over).

% taxonomic knowledge:

subsumes+(Super, Sub) :-
  subsumes(Super, Sub).

subsumes+(Super, Sub) :-
  subsumes(Super, Other-super),
  subsumes+(Other-super, Sub).
```

Anhang B

Sortierte Version von RTPLAST

Da im Moment die statische Vorvearbeitung der Sorten in RELFUN nur in der LISP-Syntax durchgeführt werden kann, ist diese Version auch in der LISP-Syntax abgebildet. Die Sorten werden durch einen geklammerten Ausdruck gekennzeichnet, wobei der Doppelpunkt als Infix notiert wird (z. B. `(_x : hostalen)`). Die relationalen Klauseln werden durch `hn` und die funktionalen durch `ft` makiert. Eine genauere Beschreibung der Syntax kann [BEH+93] entnommen werden.

```
(hn (producer (_x : hostalen) hoechst))
(hn (volume_per_unit_time_mvr_1_temperature (_x : hostalen) 190))
(hn (volume_per_unit_time_mvr_1_stress (_x : hostalen) 5))
(hn (volume_per_unit_time_mvr_2_temperature (_x : hostalen) 230))
(hn (volume_per_unit_time_mvr_2_stress (_x : hostalen) 2.16))
(hn (melting_index_mfi_1_temperature (_x : hostalen) 190))
(hn (melting_index_mfi_1_stress (_x : hostalen) 5))
(hn (melting_index_mfi_2_temperature (_x : hostalen) 230))
(hn (melting_index_mfi_2_stress (_x : hostalen) 2.16))
(hn (melting_index_mfi_3_temperature (_x : hostalen) 230))
(hn (melting_index_mfi_3_stress (_x : hostalen) 5))
(hn (recyclable (_x : hostalen) recyclable-rule))
(hn (hostalen-rest hostalen_ppk_1060_f1))
(hn (density hostalen_ppk_1060_f1 0.902))
(hn (melting_index_mfi_1 hostalen_ppk_1060_f1 2))
(hn (melting_index_mfi_2 hostalen_ppk_1060_f1 0.9))
(hn (melting_index_mfi_3 hostalen_ppk_1060_f1 4))
(hn (volume_per_unit_time_mvr_1 hostalen_ppk_1060_f1 2.3))
(hn (volume_per_unit_time_mvr_2 hostalen_ppk_1060_f1 1.3))
(hn (yield_stress hostalen_ppk_1060_f1 32))
(hn (yield_elangation hostalen_ppk_1060_f1 11))
(hn (tension_module_of_elasticity hostalen_ppk_1060_f1 1300))
(hn (tension_module_of_criep_1_h hostalen_ppk_1060_f1 850))
(hn (tension_module_of_criep_1000_h hostalen_ppk_1060_f1 500))
(hn (ball_thrust_hardness hostalen_ppk_1060_f1 67))
(hn (tension_notched-bar_impact_strength hostalen_ppk_1060_f1 48))
(hn (notched-bar_impact_strength hostalen_ppk_1060_f1 8))
(hn (izod-impact_strength_23 hostalen_ppk_1060_f1 100))
(hn (izod-impact_strength_-30 hostalen_ppk_1060_f1 14))
```

```
(hn (izod-notched-bar_impact_strength_23 hostalen_ppk_1060_f1 3.5))
(hn (izod-notched-bar_impact_strength_-30 hostalen_ppk_1060_f1 1.5))
(hn (vicat_a/50 hostalen_ppk_1060_f1 152))
(hn (vicat_b/50 hostalen_ppk_1060_f1 88))
(hn (dimensional_stability_hdt/a hostalen_ppk_1060_f1 56))
(hn (dimensional_stability_hdt/b hostalen_ppk_1060_f1 94))
(hn (used-for hostalen_ppk_1060_f1 used-for-rule))
(hn (additives hostalen_ppk_1060_f1 (tup flamability-trigger)))
(hn (density (_x : hostalen-ppn-1-prototype) 0.903))
(hn (melting_index_mfi_1 (_x : hostalen-ppn-1-prototype) 4))
(hn (melting_index_mfi_2 (_x : hostalen-ppn-1-prototype) 2))
(hn (melting_index_mfi_3 (_x : hostalen-ppn-1-prototype) 9))
(hn (volume_per_unit_time_mvr_1 (_x : hostalen-ppn-1-prototype) 4.5))
(hn (volume_per_unit_time_mvr_2 (_x : hostalen-ppn-1-prototype) 2.7))
(hn (yield_stress (_x : hostalen-ppn-1-prototype) 32))
(hn (yield_elangation (_x : hostalen-ppn-1-prototype) 11))
(hn (tension_module_of_elasticity (_x : hostalen-ppn-1-prototype) 1300) )
(hn (tension_module_of_criep_1_h (_x : hostalen-ppn-1-prototype) 900))
(hn (tension_module_of_criep_1000_h
     (_x : hostalen-ppn-1-prototype)
     480 ) )
(hn (ball_thrust_hardness (_x : hostalen-ppn-1-prototype) 68))
(hn (tension_notched-bar_impact_strength
     (_x : hostalen-ppn-1-prototype)
     45 ) )
(hn (notched-bar_impact_strength (_x : hostalen-ppn-1-prototype) 7))
(hn (izod-impact_strength_23 (_x : hostalen-ppn-1-prototype) 90))
(hn (izod-impact_strength_0 (_x : hostalen-ppn-1-prototype) 35))
(hn (izod-impact_strength_-30 (_x : hostalen-ppn-1-prototype) 12))
(hn (izod-notched-bar_impact_strength_23
     (_x : hostalen-ppn-1-prototype)
     3.1 ) )
(hn (izod-notched-bar_impact_strength_0
     (_x : hostalen-ppn-1-prototype)
     1.6 ) )
(hn (izod-notched-bar_impact_strength_-30
     (_x : hostalen-ppn-1-prototype)
     1.4 ) )
(hn (vicat_a/50 (_x : hostalen-ppn-1-prototype) 152))
(hn (vicat_b/50 (_x : hostalen-ppn-1-prototype) 89))
(hn (dimensional_stability_hdt/a (_x : hostalen-ppn-1-prototype) 56))
(hn (dimensional_stability_hdt/b (_x : hostalen-ppn-1-prototype) 90))
(hn (additives (_x : hostalen-ppn-1-prototype) (tup)))
(hn (hostalen-ppn-1 hostalen_ppn_1060))
(hn (stress-strain
     hostalen_ppn_1060
     (tup
      interpol
      (tup
       (tup 0.71 13.4)
       (tup 1.42 22.2)
       (tup 2.13 26.7)
```

```
       (tup 2.84 29.5)
       (tup 3.55 31.6)
       (tup 4.26 33.2)
       (tup 4.97 34.3)
       (tup 5.68 35.2)
       (tup 6.39 35.7)
       (tup 7.1 36.0) ) ) ) )
(hn (hostalen-ppn-1 hostalen_ppn_1060_f))
(hn (hostalen-ppn-1 hostalen_ppn_1060_f1))
(hn (hostalen-ppn-1 hostalen_ppn_1060_f3))
(hn (hostalen-ppn-1 hostalen_ppn_2060))
(hn (hostalen-ppn-1 hostalen_ppn_4160))
(hn (producer (_x : novodur) bayer))
(hn (recyclable (_x : novodur) recyclable-rule))
(hn (izod-impact_strength_23 (_x : novodur-rec-1-prototype) 60))
(hn (izod-impact_strength_-30 (_x : novodur-rec-1-prototype) 40))
(hn (izod-notched-bar_impact_strength_23
     (_x : novodur-rec-1-prototype)
     12 ) )
(hn (izod-notched-bar_impact_strength_-30
     (_x : novodur-rec-1-prototype)
     6 ) )
(hn (ball_thrust_hardness (_x : novodur-rec-1-prototype) 90))
(hn (recycled (_x : novodur-rec-1-prototype) true))
(hn (additives (_x : novodur-rec-1-prototype) (tup)))
(hn (novodur-rec-1 novodur_r_5320))
(hn (yield_stress novodur_r_5320 38))
(hn (yield_elangation novodur_r_5320 2.1))
(hn (tension_module_of_elasticity novodur_r_5320 2000))
(hn (dimensional_stability_hdt/a novodur_r_5320 90))
(hn (dimensional_stability_hdt/b novodur_r_5320 95))
(hn (novodur-rec-1 novodur_r_5322))
(hn (yield_stress novodur_r_5322 40))
(hn (yield_elangation novodur_r_5322 2.3))
(hn (tension_module_of_elasticity novodur_r_5322 2200))
(hn (dimensional_stability_hdt/a novodur_r_5322 96))
(hn (dimensional_stability_hdt/b novodur_r_5322 100))
(ft (used-for-rule hostalen_ppk_1060_f1 spritzgiessen)
    `(tup
      elektrogeraete
      technische_teile
      haushaltmaschinen
      sanitaereinrichtungen
      moebelbau
      verpackungen ) )
(ft (used-for-rule hostalen_ppk_1060_f1 pressen)
    `(tup verpackungsbaender) )
(ft (used-for-rule hostalen_ppk_1060_f1 blasformen)
    `(tup verpackungsbaender) )
(ft (recyclable-rule _plastic-id)
    (additives (_plastic-id : pp) _additives)
    (nonvar _additives)
```

```
    (member flamability-trigger _additives)
    only_in_closed_circle )
(ft (recyclable-rule _plastic-id)
    (additives (_plastic-id : pp) _additives)
    (nonvar _additives)
    (not-member flamability-trigger _additives)
    possible )
(ft (recyclable-rule _plastic-id)
    (additives (_plastic-id : aks-k) _additives)
    (nonvar _additives)
    (member flamability-trigger _additives)
    only_in_closed_circle )
(ft (recyclable-rule _plastic-id)
    (additives (_plastic-id : aks-k) _additives)
    (nonvar _additives)
    (not-member flamability-trigger _additives)
    possible )
(hn (simple-attribute identifier))
(hn (cardinality identifier (= 1)))
(hn (sort identifier string))
(hn (simple-attribute cardinality))
(hn (sort cardinality predicate))
(hn (simple-attribute sort))
(hn (cardinality sort 1))
(hn (simple-attribute method_for_test))
(hn (sort method_for_test tupel-of-strings))
(hn (simple-attribute additives))
(hn (sort additives tupel-of-strings))
(hn (simple-attribute measurement))
(hn (sort measurement tupel-of-strings))
(hn (simple-attribute exists-with-conditions))
(hn (sort exists-with-conditions tupel-of-strings))
(hn (cardinality exists-with-conditions (>= 1)))
(hn (simple-attribute recyclable))
(hn (sort recyclable boolean))
(hn (simple-attribute recycled))
(hn (sort recycled boolean))
(hn (cardinality (_x : numerical-attribute) (<= 1)))
(hn (sort (_x : numerical-attribute) real))
(hn (simple-numerical-attribute temperature))
(hn (measurement temperature c))
(hn (simple-numerical-attribute density))
(hn (method_for_test density (tup iso_1183 din_53735)))
(hn (simple-numerical-attribute volume_per_unit_time_mvr_1_temperature))
(hn (measurement volume_per_unit_time_mvr_1_temperature c))
(hn (simple-numerical-attribute volume_per_unit_time_mvr_1_stress))
(hn (measurement volume_per_unit_time_mvr_1_stress kg))
(hn (simple-numerical-attribute volume_per_unit_time_mvr_2_temperature))
(hn (measurement volume_per_unit_time_mvr_2_temperature c))
(hn (simple-numerical-attribute volume_per_unit_time_mvr_2_stress))
(hn (measurement volume_per_unit_time_mvr_2_stress kg))
(hn (simple-numerical-attribute melting_index_mfi_1_temperature))
```

```
(hn (measurement melting_index_mfi_1_temperature c))
(hn (simple-numerical-attribute melting_index_mfi_1_stress))
(hn (measurement melting_index_mfi_1_stress kg))
(hn (simple-numerical-attribute melting_index_mfi_2_temperature))
(hn (measurement melting_index_mfi_2_temperature c))
(hn (simple-numerical-attribute melting_index_mfi_2_stress))
(hn (measurement melting_index_mfi_2_stress kg))
(hn (simple-numerical-attribute melting_index_mfi_3_temperature))
(hn (measurement melting_index_mfi_3_temperature c))
(hn (simple-numerical-attribute melting_index_mfi_3_stress))
(hn (measurement melting_index_mfi_3_stress kg))
(hn (simple-numerical-attribute vicat_a/50))
(hn (method_for_test vicat_a/50 (tup din_306)))
(hn (measurement vicat_a/50 c))
(hn (simple-numerical-attribute vicat_b/50))
(hn (method_for_test vicat_b/50 (tup din_306)))
(hn (measurement vicat_b/50 c))
(hn (simple-numerical-attribute yield_elangation))
(hn (method_for_test yield_elangation (tup iso_527 din_53455)))
(hn (measurement yield_elangation proz))
(hn (simple-numerical-attribute dimensional_stability_hdt/a))
(hn (method_for_test dimensional_stability_hdt/a (tup iso_75 din_53461)))
(hn (measurement dimensional_stability_hdt/a c))
(hn (simple-numerical-attribute dimensional_stability_hdt/b))
(hn (method_for_test dimensional_stability_hdt/b (tup iso_75 din_53461)))
(hn (measurement dimensional_stability_hdt/b c))
(hn (measurement
     (_x : numerical-attribute-n/qmm-prototype)
     (/ n (^ mm 2)) ) )
(hn (numerical-attribute-n/qmm yield_stress))
(hn (method_for_test yield_stress (tup iso_527 din_53455)))
(hn (numerical-attribute-n/qmm tension_module_of_elasticity))
(hn (method_for_test tension_module_of_elasticity (tup iso_527 din_53457)))
(hn (numerical-attribute-n/qmm tension_module_of_criep_1_h))
(hn (method_for_test tension_module_of_criep_1_h (tup iso_899 din_53444)))
(hn (numerical-attribute-n/qmm tension_module_of_criep_1000_h))
(hn (method_for_test tension_module_of_criep_1000_h (tup iso_899 din_53444)))
(hn (numerical-attribute-n/qmm ball_thrust_hardness))
(hn (method_for_test ball_thrust_hardness (tup din_53456)))
(hn (measurement
     (_x : numerical-attribute-kj/qm-prototype)
     (/ kj (^ m 2)) ) )
(hn (numerical-attribute-kj/qm tension_notched-bar_impact_strength))
(hn (method_for_test
     tension_notched-bar_impact_strength
     (tup iso_8256 din_53448/1b) ) )
(hn (numerical-attribute-kj/qm notched-bar_impact_strength))
(hn (method_for_test notched-bar_impact_strength (tup din_53453)))
(hn (numerical-attribute-kj/qm izod-impact_strength_23))
(hn (method_for_test izod-impact_strength_23 (tup iso_180/1c)))
(hn (numerical-attribute-kj/qm izod-impact_strength_0))
(hn (method_for_test izod-impact_strength_0 (tup iso_180/1c)))
```

```
(hn (numerical-attribute-kj/qm izod-impact_strength_-30))
(hn (method_for_test izod-impact_strength_-30 (tup iso_180/1c)))
(hn (numerical-attribute-kj/qm izod-notched-bar_impact_strength_23))
(hn (method_for_test izod-notched-bar_impact_strength_23 (tup iso_180/1a)))
(hn (numerical-attribute-kj/qm izod-notched-bar_impact_strength_0))
(hn (method_for_test izod-notched-bar_impact_strength_0 (tup iso_180/1a)))
(hn (numerical-attribute-kj/qm izod-notched-bar_impact_strength_-30))
(hn (method_for_test izod-notched-bar_impact_strength_-30 (tup iso_180/1a)))
(hn (sort (_x : melting_index-prototype) real))
(hn (measurement (_x : melting_index-prototype) (/ g (* 10 min))))
(hn (method_for_test
     (_x : melting_index-prototype)
     (tup iso_1133 din_53735) ) )
(hn (melting_index melting_index_mfi_1))
(hn (exists-with-conditions
     melting_index_mfi_1
     (tup melting_index_mfi_1_temperature melting_index_mfi_1_stress) ) )
(hn (melting_index melting_index_mfi_2))
(hn (exists-with-conditions
     melting_index_mfi_2
     (tup melting_index_mfi_2_temperature melting_index_mfi_2_stress) ) )
(hn (melting_index melting_index_mfi_3))
(hn (exists-with-conditions
     melting_index_mfi_3
     (tup melting_index_mfi_3_temperature melting_index_mfi_3_stress) ) )
(hn (sort (_x : volume_per_unit_time-prototype) real))
(hn (measurement
     (_x : volume_per_unit_time-prototype)
     (/ (^ cm 3) (* 10 min)) ) )
(hn (method_for_test
     (_x : volume_per_unit_time-prototype)
     (tup iso_1133 din_53735) ) )
(hn (volume_per_unit_time volume_per_unit_time_mvr_1))
(hn (exists-with-conditions
     volume_per_unit_time_mvr_1
     (tup
      volume_per_unit_time_mvr_1_temperature
      volume_per_unit_time_mvr_1_stress ) ) )
(hn (volume_per_unit_time volume_per_unit_time_mvr_2))
(hn (exists-with-conditions
     volume_per_unit_time_mvr_2
     (tup
      volume_per_unit_time_mvr_2_temperature
      volume_per_unit_time_mvr_2_stress ) ) )
(hn (sort (_x : interpol-attribute) interpol-chart))
(hn (simple-interpol-attribute stress-strain))
(hn (first stress-strain dehnung))
(hn (first-measurement stress-strain proz))
(hn (second stress-strain spannung))
(hn (second-measurement stress-strain mpa))
(hn (sort (_x : rule-attribute) rule))
(hn (simple-rule-attribute
```

```
     (identifier recyclable)
     (number-of-parameter 1)
     (para identifier) ) )
(hn (simple-rule-attribute
     (identifier used-for)
     (number-of-parameter 2)
     (first-para identifier)
     (second-para processing) ) )
(ft (interpol _pairs _x)
    (sum _pairs _x _pairs 1) )
(ft (sum _pairs _x (tup) _no)
    0 )
(ft (sum _pairs _x _left-over-pairs _no)
    (is
     (tup (tup _first _second) | _left-over)
     _left-over-pairs )
    (+
     (* _second (mul _pairs _x _first _no 1))
     (sum _pairs _x _left-over (+ _no 1)) ) )
(ft (mul (tup) _x _xv _no _mulno)
    1 )
(ft (mul _pairs _x _xv _no _no)
    (is (tup id | _left-over) _pairs)
    (mul _left-over _x _xv _no (+ _no 1)) )
(ft (mul _pairs _x _xv _no _mulno)
    (is (tup (tup _first id) | _left-over) _pairs)
    (/= 0 (- _xv _first))
    (is _result (/ (- _x _first) (- _xv _first)))
    (*
     _result
     (mul _left-over _x _xv _no (+ _mulno 1)) ) )
(hn (not-member _element (tup)))
(hn (not-member _element (tup _first | _left-over))
    (string/= (princ-to-string _first) (princ-to-string _element))
    (not-member _element _left-over) )
(hn (subsumes+ _super _sub)
    (subsumes _super _sub) )
(hn (subsumes+ _super _sub)
    (subsumes _super _other-super)
    (subsumes+ _other-super _sub) )
(hn (plastic _x)
    (mechanical-property _x) )
(hn (plastic _x)
    (general-property _x) )
(hn (plastic _x)
    (property-by-method _x) )
(hn (plastic _x)
    (property-by-rule _x) )
(hn (general-property _x)
    (thermoplastic _x) )
(hn (mechanical-property _x)
    (thermoplastic _x) )
```

```
(hn (property-by-method _x)
    (thermoplastic _x) )
(hn (property-by-rule _x)
    (thermoplastic _x) )
(hn (thermoplastic _x)
    (pp _x) )
(hn (thermoplastic _x)
    (absc _x) )
(hn (pp _x)
    (hostalen _x) )
(hn (hostalen _x)
    (hostalen-rest _x) )
(hn (hostalen _x)
    (hostalen-ppn-1-prototype _x) )
(hn (hostalen-ppn-1-prototype _x)
    (hostalen-ppn-1 _x) )
(hn (absc _x)
    (novodur _x) )
(hn (novodur _x)
    (novodur-rec-1-prototype _x) )
(hn (novodur-rec-1-prototype _x)
    (novodur-rec-1 _x) )
(hn (plastic _x)
    (attribute _x) )
(hn (attribute _x)
    (simple-attribute _x) )
(hn (attribute _x)
    (numerical-attribute _x) )
(hn (numerical-attribute _x)
    (simple-numerical-attribute _x) )
(hn (numerical-attribute _x)
    (numerical-attribute-n/qmm-prototype _x) )
(hn (numerical-attribute-n/qmm-prototype _x)
    (numerical-attribute-n/qmm _x) )
(hn (numerical-attribute _x)
    (numerical-attribute-kj/qm-prototype _x) )
(hn (numerical-attribute-kj/qm-prototype _x)
    (numerical-attribute-kj/qm _x) )
(hn (numerical-attribute _x)
    (numerical-attribute-with-conditions _x) )
(hn (numerical-attribute-with-conditions _x)
    (melting_index-prototype _x) )
(hn (melting_index-prototype _x)
    (melting_index _x) )
(hn (numerical-attribute-with-conditions _x)
    (volume_per_unit_time-prototype _x) )
(hn (volume_per_unit_time-prototype _x)
    (volume_per_unit_time _x) )
(hn (attribute _x)
    (interpol-attribute _x) )
(hn (interpol-attribute _x)
    (simple-interpol-attribute _x) )
```

```
(hn (attribute _x)
    (rule-attribute _x) )
(hn (rule-attribute _x)
    (simple-rule-attribute-1 _x) )
(hn (rule-attribute _x)
    (simple-rule-attribute-2 _x) )
```

Anhang C

Beispieldialog zu RTPLAST

Der folgende Text ist ein Beispieldialog zu RTPLAST im RELFUN-System. Die objektzentrierte und die sortierte Repräsentation werden an geeigneten Stellen verglichen.

```
rfi-1> exec rtplast

rfi-1> style prolog
rfi-p>
rfi-p> %------------------------------------------------------------------------
rfi-p> %|      A Declarative Knowledge Base on Recyclable Thermoplastics       |
rfi-p> %|       Using Attribute Terms, Sorts, and Inheritance in RELFUN        |
rfi-p> %------------------------------------------------------------------------
rfi-p>
rfi-p> %                                                        April 1994
rfi-p> % (c) Ulrich Buhrmann, Michael Sintek
rfi-p> %
rfi-p> emul
rfe-p> replace rtplast
% Reading file "/home/buhrmann/rfm/sampler/kunststoffe/rtplast.rfp" ..
rfe-p> % uses the n-ary translation:
rfe-p> orf n
rfe-p> size
(clauses 136)
(literals 178)
rfe-p> compile
rfe-p> size
(clauses 138)
(literals 466)
rfe-p>
rfe-p>
rfe-p> %           Original object-centered version of RTPLAST
rfe-p> %------------------------------------------------------------------------
rfe-p>
rfe-p> % Two sample instances of one sample class of recyclable novodurs
rfe-p> % (attribute-value pairs prefixed by the class name)
rfe-p> % novodur-rec-1(
```

```
rfe-p> %   identifier[novodur_r_5320],
rfe-p> %   yield_stress[38],
rfe-p> %   yield_elangation[2.1],
rfe-p> %   tension_module_of_elasticity[2000],
rfe-p> %   dimensional_stability_hdt/a[90],
rfe-p> %   dimensional_stability_hdt/b[95]).
rfe-p> % novodur-rec-1(
rfe-p> %   identifier[novodur_r_5322],
rfe-p> %   yield_stress[40],
rfe-p> %   yield_elangation[2.3],
rfe-p> %   tension_module_of_elasticity[2200],
rfe-p> %   dimensional_stability_hdt/a[96],
rfe-p> %   dimensional_stability_hdt/b[100]).
rfe-p>
rfe-p> % Goal retrieving the elasticity modules of both instances
rfe-p> novodur-rec-1(identifier[I], tension_module_of_elasticity[E-module])
true
I = novodur_r_5320
E-module = 2000
rfe-p> more
true
I = novodur_r_5322
E-module = 2200
rfe-p> more
unknown
rfe-p> inter
rfi-p> pause()

rfi-p> bye
true
rfi-p> emul
rfe-p>
rfe-p> % One sample prototype declared below novodur and above novodur-rec-1
rfe-p> % declare(proto-class[novodur-rec-1-prototype,
rfe-p> %                     super[novodur]]).
rfe-p> %
rfe-p> % novodur-rec-1-prototype(
rfe-p> %   izod-impact_strength_23[60],
rfe-p> %   izod-impact_strength_-30[40],
rfe-p> %   izod-notched-bar_impact_strength_23[12],
rfe-p> %   izod-notched-bar_impact_strength_-30[6],
rfe-p> %   ball_thrust_hardness[90],
rfe-p> %   recycled[true],
rfe-p> %   additives[[]]).
rfe-p> %
rfe-p> % declare(indi-class[novodur-rec-1,
rfe-p> %                    super[novodur-rec-1-prototype]]).
rfe-p>
rfe-p> % Goal additionally inheriting the ball thrust hardness from the prototype
rfe-p> novodur-rec-1(identifier[I],
                     tension_module_of_elasticity[E-module],
                     ball_thrust_hardness[Bh])
```

```
true
I = novodur_r_5320
E-module = 2000
Bh = 90
rfe-p> more
true
I = novodur_r_5322
E-module = 2200
Bh = 90
rfe-p> inter
rfi-p> pause()

rfi-p> bye
true
rfi-p> emul
rfe-p>
rfe-p> % Querying the same attributes for all instances below thermoplastic
rfe-p> % (the variable Indi-class will be bound to their class names)
rfe-p> instance>(thermoplastic(),
                 Indi-class(identifier[I],
                            tension_module_of_elasticity[E-module],
                            ball_thrust_hardness[Bh]))
true
Indi-class = hostalen-rest
I = hostalen_ppk_1060_f1
E-module = 1300
Bh = 67
rfe-p> more
true
Indi-class = hostalen-ppn-1
I = hostalen_ppn_1060
E-module = 1300
Bh = 68
rfe-p> more
true
Indi-class = hostalen-ppn-1
I = hostalen_ppn_1060_f
E-module = 1300
Bh = 68
rfe-p> more
true
Indi-class = hostalen-ppn-1
I = hostalen_ppn_1060_f1
E-module = 1300
Bh = 68
rfe-p> more
true
Indi-class = hostalen-ppn-1
I = hostalen_ppn_1060_f3
E-module = 1300
Bh = 68
rfe-p> more
```

```
true
Indi-class = hostalen-ppn-1
I = hostalen_ppn_2060
E-module = 1300
Bh = 68
rfe-p> more
true
Indi-class = hostalen-ppn-1
I = hostalen_ppn_4160
E-module = 1300
Bh = 68
rfe-p> more
true
Indi-class = novodur-rec-1
I = novodur_r_5320
E-module = 2000
Bh = 90
rfe-p> more
true
Indi-class = novodur-rec-1
I = novodur_r_5322
E-module = 2200
Bh = 90
rfe-p> inter
rfi-p> pause()

rfi-p> bye
true
rfi-p> emul
rfe-p>
rfe-p> % Conjunction implementing simple material pre-selection
rfe-p> % (P1="ball thrust hardness greater 85",
rfe-p> %  M1=novodur_r_5320, M2=novodur_r_5322)
rfe-p> instance>(thermoplastic(), Indi-class(identifier[I],
                                              ball_thrust_hardness[Bh])),
       nonvar(Bh), >(Bh, 85)
t
Indi-class = novodur-rec-1
I = novodur_r_5320
Bh = 90
rfe-p> more
t
Indi-class = novodur-rec-1
I = novodur_r_5322
Bh = 90
rfe-p> more
unknown
rfe-p> inter
rfi-p> pause()

rfi-p> bye
true
```

```
rfi-p> emul
rfe-p>
rfe-p> % cost for novodur_r_5320 and novodur_r_5322 added Horn-logically
rfe-p> l cost
cost(novodur_r_5320, 4.3).
cost(novodur_r_5322, 4.5).
rfe-p>
rfe-p> % Goal implementing special material selection
rfe-p> % (optimality criterion assumed to be cost minimization)
rfe-p> min-cost([novodur_r_5320, novodur_r_5322], M)
true
M = novodur_r_5320
rfe-p>
rfe-p> pause()
unknown
rfe-p>
rfe-p> % Goal implementing the interpolation of the attribute value of
rfe-p> % stress-strain by a higher order function call
rfe-p> instance>(thermoplastic(),
                 Indi-class(identifier[I],
                            stress-strain[[Method, Chart]])),
       nonvar(Method),
       Method(Chart,2.3)
27.473159400564154
Indi-class = hostalen-ppn-1
I = hostalen_ppn_1060
Method = interpol
Chart = [[0.71, 13.4],
         [1.42, 22.2],
         [2.13, 26.7],
         [2.84, 29.5],
         [3.55, 31.6],
         [4.26, 33.2],
         [4.97, 34.3],
         [5.68, 35.2],
         [6.39, 35.7],
         [7.1, 36.0]]
rfe-p> inter
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> %               working with the positioned representation
rfi-p> %-------------------------------------------------------------------------
rfi-p>
rfi-p> replace rtplast
% Reading file "/home/buhrmann/rfm/sampler/kunststoffe/rtplast.rfp" ..
rfi-p> orf n
rfi-p> undeclare
rfi-p> untype
```

```
rfi-p> normalize
rfi-p>
rfi-p> listing novodur-rec-1
novodur-rec-1(Additives1989,
              Ball_thrust_hardness1990,
              Density1991,
              90,
              95,
              novodur_r_5320,
              Izod-impact_strength_-301995,
              Izod-impact_strength_01996,
              Izod-impact_strength_231997,
              Izod-notched-bar_impact_strength_-301998,
              Izod-notched-bar_impact_strength_01999,
              Izod-notched-bar_impact_strength_232000,
              Melting_index_mfi_12001,
              Melting_index_mfi_1_stress2002,
              Melting_index_mfi_1_temperature2003,
              Melting_index_mfi_22004,
              Melting_index_mfi_2_stress2005,
              Melting_index_mfi_2_temperature2006,
              Melting_index_mfi_32007,
              Melting_index_mfi_3_stress2008,
              Melting_index_mfi_3_temperature2009,
              Notched-bar_impact_strength2010,
              Producer2011,
              Recyclable2012,
              Recycled2013,
              Stress-strain2014,
              Tension_module_of_criep_1000_h2015,
              Tension_module_of_criep_1_h2016,
              2000,
              Tension_notched-bar_impact_strength2018,
              Used-for2019,
              Vicat_a/502020,
              Vicat_b/502021,
              Volume_per_unit_time_mvr_12022,
              Volume_per_unit_time_mvr_1_stress2023,
              Volume_per_unit_time_mvr_1_temperature2024,
              Volume_per_unit_time_mvr_22025,
              Volume_per_unit_time_mvr_2_stress2026,
              Volume_per_unit_time_mvr_2_temperature2027,
              2.1,
              38)
    :- novodur-rec-1-prototype(Additives1989,
                               Ball_thrust_hardness1990,
                               Density1991,
                               90,
                               95,
                               novodur_r_5320,
                               Izod-impact_strength_-301995,
                               Izod-impact_strength_01996,
```

```
                                Izod-impact_strength_231997,
                                Izod-notched-bar_impact_strength_-301998,
                                Izod-notched-bar_impact_strength_01999,
                                Izod-notched-bar_impact_strength_232000,
                                Melting_index_mfi_12001,
                                Melting_index_mfi_1_stress2002,
                                Melting_index_mfi_1_temperature2003,
                                Melting_index_mfi_22004,
                                Melting_index_mfi_2_stress2005,
                                Melting_index_mfi_2_temperature2006,
                                Melting_index_mfi_32007,
                                Melting_index_mfi_3_stress2008,
                                Melting_index_mfi_3_temperature2009,
                                Notched-bar_impact_strength2010,
                                Producer2011,
                                Recyclable2012,
                                Recycled2013,
                                Stress-strain2014,
                                Tension_module_of_criep_1000_h2015,
                                Tension_module_of_criep_1_h2016,
                                2000,
                                Tension_notched-bar_impact_strength2018,
                                Used-for2019,
                                Vicat_a/502020,
                                Vicat_b/502021,
                                Volume_per_unit_time_mvr_12022,
                                Volume_per_unit_time_mvr_1_stress2023,
                                Volume_per_unit_time_mvr_1_temperature2024,
                                Volume_per_unit_time_mvr_22025,
                                Volume_per_unit_time_mvr_2_stress2026,
                                Volume_per_unit_time_mvr_2_temperature2027,
                                2.1,
                                38).
novodur-rec-1(Additives2030,
              Ball_thrust_hardness2031,
              Density2032,
              96,
              100,
              novodur_r_5322,
              Izod-impact_strength_-302036,
              Izod-impact_strength_02037,
              Izod-impact_strength_232038,
              Izod-notched-bar_impact_strength_-302039,
              Izod-notched-bar_impact_strength_02040,
              Izod-notched-bar_impact_strength_232041,
              Melting_index_mfi_12042,
              Melting_index_mfi_1_stress2043,
              Melting_index_mfi_1_temperature2044,
              Melting_index_mfi_22045,
              Melting_index_mfi_2_stress2046,
              Melting_index_mfi_2_temperature2047,
              Melting_index_mfi_32048,
```

```
          Melting_index_mfi_3_stress2049,
          Melting_index_mfi_3_temperature2050,
          Notched-bar_impact_strength2051,
          Producer2052,
          Recyclable2053,
          Recycled2054,
          Stress-strain2055,
          Tension_module_of_criep_1000_h2056,
          Tension_module_of_criep_1_h2057,
          2200,
          Tension_notched-bar_impact_strength2059,
          Used-for2060,
          Vicat_a/502061,
          Vicat_b/502062,
          Volume_per_unit_time_mvr_12063,
          Volume_per_unit_time_mvr_1_stress2064,
          Volume_per_unit_time_mvr_1_temperature2065,
          Volume_per_unit_time_mvr_22066,
          Volume_per_unit_time_mvr_2_stress2067,
          Volume_per_unit_time_mvr_2_temperature2068,
          2.3,
          40)
:- novodur-rec-1-prototype(Additives2030,
                           Ball_thrust_hardness2031,
                           Density2032,
                           96,
                           100,
                           novodur_r_5322,
                           Izod-impact_strength_-302036,
                           Izod-impact_strength_02037,
                           Izod-impact_strength_232038,
                           Izod-notched-bar_impact_strength_-302039,
                           Izod-notched-bar_impact_strength_02040,
                           Izod-notched-bar_impact_strength_232041,
                           Melting_index_mfi_12042,
                           Melting_index_mfi_1_stress2043,
                           Melting_index_mfi_1_temperature2044,
                           Melting_index_mfi_22045,
                           Melting_index_mfi_2_stress2046,
                           Melting_index_mfi_2_temperature2047,
                           Melting_index_mfi_32048,
                           Melting_index_mfi_3_stress2049,
                           Melting_index_mfi_3_temperature2050,
                           Notched-bar_impact_strength2051,
                           Producer2052,
                           Recyclable2053,
                           Recycled2054,
                           Stress-strain2055,
                           Tension_module_of_criep_1000_h2056,
                           Tension_module_of_criep_1_h2057,
                           2200,
                           Tension_notched-bar_impact_strength2059,
```

```
                                Used-for2060,
                                Vicat_a/502061,
                                Vicat_b/502062,
                                Volume_per_unit_time_mvr_12063,
                                Volume_per_unit_time_mvr_1_stress2064,
                                Volume_per_unit_time_mvr_1_temperature2065,
                                Volume_per_unit_time_mvr_22066,
                                Volume_per_unit_time_mvr_2_stress2067,
                                Volume_per_unit_time_mvr_2_temperature2068,
                                2.3,
                                40).
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> novodur-rec-1(| All)
true
All = [[],
       90,
       Density1391*6,
       90,
       95,
       novodur_r_5320,
       40,
       Izod-impact_strength_01376*8,
       60,
       6,
       Izod-notched-bar_impact_strength_01379*8,
       12,
       Melting_index_mfi_11393*6,
       Melting_index_mfi_1_stress1394*6,
       Melting_index_mfi_1_temperature1395*6,
       Melting_index_mfi_21396*6,
       Melting_index_mfi_2_stress1397*6,
       Melting_index_mfi_2_temperature1398*6,
       Melting_index_mfi_31399*6,
       Melting_index_mfi_3_stress1400*6,
       Melting_index_mfi_3_temperature1401*6,
       Notched-bar_impact_strength1381*8,
       bayer,
       recyclable-rule,
       true,
       Stress-strain1411*10,
       Tension_module_of_criep_1000_h1382*8,
       Tension_module_of_criep_1_h1383*8,
       2000,
       Tension_notched-bar_impact_strength1385*8,
       Used-for1414*12,
       Vicat_a/501386*8,
       Vicat_b/501387*8,
```

```
       Volume_per_unit_time_mvr_11404*6,
       Volume_per_unit_time_mvr_1_stress1405*6,
       Volume_per_unit_time_mvr_1_temperature1406*6,
       Volume_per_unit_time_mvr_21407*6,
       Volume_per_unit_time_mvr_2_stress1408*6,
       Volume_per_unit_time_mvr_2_temperature1409*6,
       2.1,
       38]
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> novodur-rec-1(_, Ball_thrust_hardness, _, _, _, Identifier | _)
true
Ball_thrust_hardness = 90
Identifier = novodur_r_5320
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> trace novodur-rec-1 novodur-rec-1-prototype
rfi-p> novodur-rec-1(_, Ball_thrust_hardness, _, _, _, Identifier | _)
>1  novodur-rec-1(Additives1989*1,
                  Ball_thrust_hardness1990*1,
                  Density1991*1,
                  90,
                  95,
                  novodur_r_5320,
                  Izod-impact_strength_-301995*1,
                  Izod-impact_strength_01996*1,
                  Izod-impact_strength_231997*1,
                  Izod-notched-bar_impact_strength_-301998*1,
                  Izod-notched-bar_impact_strength_01999*1,
                  Izod-notched-bar_impact_strength_232000*1,
                  Melting_index_mfi_12001*1,
                  Melting_index_mfi_1_stress2002*1,
                  Melting_index_mfi_1_temperature2003*1,
                  Melting_index_mfi_22004*1,
                  Melting_index_mfi_2_stress2005*1,
                  Melting_index_mfi_2_temperature2006*1,
                  Melting_index_mfi_32007*1,
                  Melting_index_mfi_3_stress2008*1,
                  Melting_index_mfi_3_temperature2009*1,
                  Notched-bar_impact_strength2010*1,
                  Producer2011*1,
                  Recyclable2012*1,
                  Recycled2013*1,
                  Stress-strain2014*1,
```

```
                Tension_module_of_criep_1000_h2015*1,
                Tension_module_of_criep_1_h2016*1,
                2000,
                Tension_notched-bar_impact_strength2018*1,
                Used-for2019*1,
                Vicat_a/502020*1,
                Vicat_b/502021*1,
                Volume_per_unit_time_mvr_12022*1,
                Volume_per_unit_time_mvr_1_stress2023*1,
                Volume_per_unit_time_mvr_1_temperature2024*1,
                Volume_per_unit_time_mvr_22025*1,
                Volume_per_unit_time_mvr_2_stress2026*1,
                Volume_per_unit_time_mvr_2_temperature2027*1,
                2.1,
                38) :-
>1  novodur-rec-1-prototype(Additives1948*9,
                           90,
                           Density1950*9,
                           90,
                           95,
                           novodur_r_5320,
                           40,
                           Izod-impact_strength_01955*9,
                           60,
                           6,
                           Izod-notched-bar_impact_strength_01958*9,
                           12,
                           Melting_index_mfi_11960*9,
                           Melting_index_mfi_1_stress1961*9,
                           Melting_index_mfi_1_temperature1962*9,
                           Melting_index_mfi_21963*9,
                           Melting_index_mfi_2_stress1964*9,
                           Melting_index_mfi_2_temperature1965*9,
                           Melting_index_mfi_31966*9,
                           Melting_index_mfi_3_stress1967*9,
                           Melting_index_mfi_3_temperature1968*9,
                           Notched-bar_impact_strength1969*9,
                           Producer1970*9,
                           Recyclable1971*9,
                           true,
                           Stress-strain1973*9,
                           Tension_module_of_criep_1000_h1974*9,
                           Tension_module_of_criep_1_h1975*9,
                           2000,
                           Tension_notched-bar_impact_strength1977*9,
                           Used-for1978*9,
                           Vicat_a/501979*9,
                           Vicat_b/501980*9,
                           Volume_per_unit_time_mvr_11981*9,
                           Volume_per_unit_time_mvr_1_stress1982*9,
                           Volume_per_unit_time_mvr_1_temperature1983*9,
                           Volume_per_unit_time_mvr_21984*9,
```

```
                              Volume_per_unit_time_mvr_2_stress1985*9,
                              Volume_per_unit_time_mvr_2_temperature1986*9,
                              2.1,
                              38) :-
  <1  novodur-rec-1-prototype([],
                              90,
                              Density1391*20,
                              90,
                              95,
                              novodur_r_5320,
                              40,
                              Izod-impact_strength_01376*22,
                              60,
                              6,
                              Izod-notched-bar_impact_strength_01379*22,
                              12,
                              Melting_index_mfi_11393*20,
                              Melting_index_mfi_1_stress1394*20,
                              Melting_index_mfi_1_temperature1395*20,
                              Melting_index_mfi_21396*20,
                              Melting_index_mfi_2_stress1397*20,
                              Melting_index_mfi_2_temperature1398*20,
                              Melting_index_mfi_31399*20,
                              Melting_index_mfi_3_stress1400*20,
                              Melting_index_mfi_3_temperature1401*20,
                              Notched-bar_impact_strength1381*22,
                              bayer,
                              recyclable-rule,
                              true,
                              Stress-strain1411*24,
                              Tension_module_of_criep_1000_h1382*22,
                              Tension_module_of_criep_1_h1383*22,
                              2000,
                              Tension_notched-bar_impact_strength1385*22,
                              Used-for1414*26,
                              Vicat_a/501386*22,
                              Vicat_b/501387*22,
                              Volume_per_unit_time_mvr_11404*20,
                              Volume_per_unit_time_mvr_1_stress1405*20,
                              Volume_per_unit_time_mvr_1_temperature1406*20,
                              Volume_per_unit_time_mvr_21407*20,
                              Volume_per_unit_time_mvr_2_stress1408*20,
                              Volume_per_unit_time_mvr_2_temperature1409*20,
                              2.1,
                              38) :-
<1  novodur-rec-1([],
                  90,
                  Density1391*20,
                  90,
                  95,
                  novodur_r_5320,
                  40,
```

```
                  Izod-impact_strength_01376*22,
                  60,
                  6,
                  Izod-notched-bar_impact_strength_01379*22,
                  12,
                  Melting_index_mfi_11393*20,
                  Melting_index_mfi_1_stress1394*20,
                  Melting_index_mfi_1_temperature1395*20,
                  Melting_index_mfi_21396*20,
                  Melting_index_mfi_2_stress1397*20,
                  Melting_index_mfi_2_temperature1398*20,
                  Melting_index_mfi_31399*20,
                  Melting_index_mfi_3_stress1400*20,
                  Melting_index_mfi_3_temperature1401*20,
                  Notched-bar_impact_strength1381*22,
                  bayer,
                  recyclable-rule,
                  true,
                  Stress-strain1411*24,
                  Tension_module_of_criep_1000_h1382*22,
                  Tension_module_of_criep_1_h1383*22,
                  2000,
                  Tension_notched-bar_impact_strength1385*22,
                  Used-for1414*26,
                  Vicat_a/501386*22,
                  Vicat_b/501387*22,
                  Volume_per_unit_time_mvr_11404*20,
                  Volume_per_unit_time_mvr_1_stress1405*20,
                  Volume_per_unit_time_mvr_1_temperature1406*20,
                  Volume_per_unit_time_mvr_21407*20,
                  Volume_per_unit_time_mvr_2_stress1408*20,
                  Volume_per_unit_time_mvr_2_temperature1409*20,
                  2.1,
                  38) :-
true
Ball_thrust_hardness = 90
Identifier = novodur_r_5320
rfi-p> nospy
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p>
rfi-p> %          Translated sorted Horn-logic version of RTPLAST
rfi-p> %-------------------------------------------------------------------------
rfi-p>
rfi-p> replace rtplast
% Reading file "/home/buhrmann/rfm/sampler/kunststoffe/rtplast.rfp" ..
rfi-p> % uses the sorted version:
rfi-p> orf 2
```

```
rfi-p> size
(clauses 136)
(literals 178)
rfi-p> undeclare
rfi-p> unorf
rfi-p> size
(clauses 265)
(literals 342)
rfi-p> inter
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> % listing of one sample class of recyclable novodurs
rfi-p> % (class name becomes unary predicate)
rfi-p> l novodur-rec-1
novodur-rec-1(novodur_r_5320).
novodur-rec-1(novodur_r_5322).
rfi-p>
rfi-p> % 'object centered' LISTINGS of the sample instances novodur_r_5320
rfi-p> % and novodur_r_5322
rfi-p> % (attributes become binary predicates, instances copied into first
rfi-p> %  argument)
rfi-p> l Attribute(novodur_r_5320, Value)
yield_stress(novodur_r_5320, 38).
yield_elangation(novodur_r_5320, 2.1).
tension_module_of_elasticity(novodur_r_5320, 2000).
dimensional_stability_hdt/a(novodur_r_5320, 90).
dimensional_stability_hdt/b(novodur_r_5320, 95).
cost(novodur_r_5320, 4.3).
rfi-p> l Attribute(novodur_r_5322, Value)
yield_stress(novodur_r_5322, 40).
yield_elangation(novodur_r_5322, 2.3).
tension_module_of_elasticity(novodur_r_5322, 2200).
dimensional_stability_hdt/a(novodur_r_5322, 96).
dimensional_stability_hdt/b(novodur_r_5322, 100).
cost(novodur_r_5322, 4.5).
rfi-p>
rfi-p> % 'attribute centered' LISTING of tension_module_of_elasticity
rfi-p> % (':' associates the sort hostalen-ppn-1-prototype with the Variable X)
rfi-p> l tension_module_of_elasticity
tension_module_of_elasticity(hostalen_ppk_1060_f1, 1300).
tension_module_of_elasticity(X : hostalen-ppn-1-prototype, 1300).
tension_module_of_elasticity(novodur_r_5320, 2000).
tension_module_of_elasticity(novodur_r_5322, 2200).
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
```

```
rfi-p> % meta-information about the attribute dimensional_stability_hdt/a
rfi-p> l Attribute(dimensional_stability_hdt/a, Value)
method_for_test(dimensional_stability_hdt/a, [iso_75, din_53461]).
measurement(dimensional_stability_hdt/a, c).
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> untype
rfi-p> uncomma
rfi-p> size
(clauses 265)
(literals 404)
rfi-p>
rfi-p> % listing of tension_module_of_elasticity after transformation of
rfi-p> % ':' sorts to unary predicates
rfi-p> l tension_module_of_elasticity
tension_module_of_elasticity(hostalen_ppk_1060_f1, 1300).
tension_module_of_elasticity(X, 1300) :- hostalen-ppn-1-prototype(X).
tension_module_of_elasticity(novodur_r_5320, 2000).
tension_module_of_elasticity(novodur_r_5322, 2200).
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> % Earlier object-centered query
rfi-p> % novodur-rec-1(identifier[I], tension_module_of_elasticity[E-module])
rfi-p> % becomes equivalent Ident-conjoined Horn query
rfi-p> novodur-rec-1(Ident), tension_module_of_elasticity(Ident, E-module)
true
Ident = novodur_r_5320
E-module = 2000
rfi-p> more
true
Ident = novodur_r_5322
E-module = 2200
rfi-p> more
unknown
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> % The sample protoype Horn-rule-defined above novodur-rec-1
rfi-p> l novodur-rec-1-prototype
novodur-rec-1-prototype(X) :- novodur-rec-1(X).
rfi-p>
```

```
rfi-p> % ‘attribute-centered’ listing shows ball_thrust_hardness of the
rfi-p> % novodur prototype and of other plastics
rfi-p> l ball_thrust_hardness
ball_thrust_hardness(hostalen_ppk_1060_f1, 67).
ball_thrust_hardness(X, 68) :- hostalen-ppn-1-prototype(X).
ball_thrust_hardness(X, 90) :- novodur-rec-1-prototype(X).
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> % Earlier object-centered, inheriting goal
rfi-p> % novodur-rec-1(identifier[I],
rfi-p> %               tension_module_of_elasticity[E-module],
rfi-p> %               ball_thrust_hardness[Bh])
rfi-p> % becomes equivalent I-conjoined Horn query, inheriting via the above rules
rfi-p> novodur-rec-1(I),
       tension_module_of_elasticity(I, E-module),
       ball_thrust_hardness(I, Bh)
true
I = novodur_r_5320
E-module = 2000
Bh = 90
rfi-p> more
true
I = novodur_r_5322
E-module = 2200
Bh = 90
rfi-p> more
unknown
rfi-p>
rfi-p> pause()

rfi-p> bye
true
rfi-p>
rfi-p> % Query of all instances below thermoplastic with fixed ball_thrust_hardness
rfi-p> %  instance>(thermoplastic(),
rfi-p> %            Indi-class(identifier[I],
rfi-p> %                       tension_module_of_elasticity[E-module],
rfi-p> %                       ball_thrust_hardness[90]))
rfi-p> % becomes Horn conjunction starting with thermoplastic predicate
rfi-p> % (no Indi-class variable)
rfi-p> thermoplastic(I),
       tension_module_of_elasticity(I, E-module),
       ball_thrust_hardness(I, 90)
true
I = novodur_r_5320
E-module = 2000
rfi-p> more
true
```

```
I = novodur_r_5322
E-module = 2200
rfi-p> more
unknown
rfi-p>
```

Anhang D

Glossar

ABSC Acrylonitrile Butadiene Styrene Copolymer

Additive additives

Dichte density

DFKI Deutsches Forschungszentrum für Künstliche Intelligenz

Evolution Oberbegriff für Exploration und Validierung

Formbeständigkeit dimensional stability

GMT glasmattenverstärkter Thermoplast

IVW Institut für Verbundwerkstoffe

Izod-Schlagzähigkeit Izod-impact strength

KB Knowledge Base

Kerbschlagzähigkeit notched-bar impact strength

Kugeldruckhärte ball thrust hardness

Kunststoff plastic

ORF object-centered RELFUN

PP Polypropylen

RPPP recyclinggerechte Produkt- und Produktionsplanung

RTPLAST Wissensbasis über recyclierbare Thermoplaste

Schmelzindex melting index, index of melting

Spannungsdehnungsdiagramm stress-strain-diagramm

Streckdehnung yield elangation

Streckspannung yield stress

Thermoplaste thermoplastics

VEGA Knowledge Validation and Exploration by Global Analysis

Verbundwerkstoffe composite materials (composites)

Volumenfließrate volume per unit time

Werkstoffverbunde composite structures (macro composites)

Zug-E-Modul tension module of elasticity

Zug-Kriechmodul tension module of criep

Zugkerbschlagzähigkeit tension notched-bar impact strength

Das hier abgedruckte Werkstoffvokabular ist einem von Dirk Blum erstelleten Glossar entnommen, wofür ich mich an dieser Stelle bei ihm bedanken möchte.