Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-93-46

A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning

Philipp Hanschke

Oktober 1993

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, SEMA Group, and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

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

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

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Friedrich J. Wendl
Director

A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning

Philipp Hanschke

DFKI-RR-93-46

This report contains the essentials of the dissertation of the author at the University of Kaiserslautern and the theoretical foundations of the terminological reasoning system TAXON. Chapter 3 is a deeply revised version of DFKI Report RR-92-37, which has also been published in the proceedings of KR'92.

This work has been supported by grants from The Federal Ministry for Research and Technology (FKZ ITW 8902 C4 and FKZ 413 5839 ITW 9304/3).

A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning

Philipp Hanschke

Abstract

The paper settles a research branch in the realm of logic-oriented, hybrid knowledge representation. Terminological knowledge representation and reasoning can now be utilized for more realistic applications as an integral component of a computationally complete, declarative hybrid knowledge representation formalism with integrated special-purpose reasoners of concrete domains such as real-closed fields or finite-domain constraints. The paper presents technical results exploring the impact of "role interaction" on the decidability of the subsumption problem of terminological logics. In particular, decision procedures are presented for common reasoning problems in an expressive terminological logic that is parametrized by a concrete domain. A refined minimal belief logic which avoids certain problems concerning the non-propositional case (which occurred surprisingly) is the basis of the model-theoretic semantics of a very general generic rule formalism integrating goal-directed (i.e., top-down) and data-driven (i.e., bottom-up) reasoning in a declarative manner. A mechanical engineering application (production planning of lathes) is used to demonstrate how the theoretical results can be employed in realistic applications.

Contents

List of Figures

Chapter 1

Introduction

In his invited talk at the 8^{th} National Conference on AI in 1990 [Brachman, 1990] Ron Brachman argued that the development of "unified reasoners" is one potential highlight of the "future of knowledge representation" (p. 1089). Similarly, "incomplete reasoners" and "expressiveness vs. tractability" are mentioned in a list of important open research problems (p. 1090). This thesis contributes to all of these issues. In particular, research on "unified reasoners" in a subfield of symbolic, logic-oriented knowledge representation has been settled to a certain extent. Terminological knowledge representation and reasoning can now be utilized for more realistic applications as an integral component of a computationally complete, declarative hybrid knowledge representation formalism with integrated special-purpose reasoners of concrete domains such as real-closed fields or finite-domain constraints.

Concept languages based on KL-ONE are mostly used to represent the terminological knowledge of a particular problem domain on an abstract logical level. In this thesis some extensions of terminological formalisms serving demands from realistic applications are considered and analyzed with respect to decidability of inference services, which is a contribution to the issue of "expressiveness vs. tractability" in the area of terminological knowledge representation. In particular, Chapter 3 provides an overview of concept forming operators dealing with role interaction, such as the well known role-value maps.

Some important representation and reasoning demands can be satisfied by an enhanced, generic terminological formalism which is introduced in Chapter 3 and is parametrized by a *concrete domain* (such as real numbers). However, the trade-off between expressiveness and tractability imposes significant limitations on the expressiveness for terminological formalisms (which have decidable reasoning problems associated with their inference services).

To overcome these limitations a generic, semidecidable rule scheme integrating data-driven (e.g., production rules) and goal-directed (à la Prolog) reasoning is developed. Even at this scheme level it is shown how inference algorithms can be constructed from the reasoning algorithms of the chosen underlying *condition logic*, which is a first-order language satisfying certain requirements. It is also shown at

this scheme level that the inferences are sound and complete with respect to a minimal belief logic. This logic had to be modified to become a 'really' minimal belief logic, also in the first-order case. This characterization of the operational semantics (which is incomplete with respect to a reading of the rules in classical logic) by model-theoretic means is a contribution to the issue of "incomplete reasoners".

If the generic terminological formalism is instantiated with an *admissible concrete domain*, and the resulting formalism is then inserted into the rule scheme, a declarative, hybrid knowledge representation formalism is obtained that integrates four essential reasoning paradigms: terminological, constraint-based (in the concrete domain), data-driven, and goal-directed. Thus, with these two schemes powerful "unified reasoners" for symbolic, logic-oriented knowledge representation can be constructed that are accompanied by a declarative semantics.

Taking a knowledge engineering point of view, the generic hybrid formalism can be described in terms of three layers.

Concrete Domains: This layer provides access to tuned, efficient reasoning algorithms of well understood (concrete) domains.

Terminological Formalism: On this layer a vocabulary tailored to the application domain and grounded in the concrete domains can be developed. The services of the terminological formalism aid the knowledge engineer in analyzing his definitions and minimizing the number of errors.

Rules: On this layer the actual knowledge for problem solving can be represented on the basis of a "save" vocabulary and efficient special-purpose reasoning algorithms contained in the lower layers.

This perspective reveals that terminological reasoning can be fully integrated in a knowledge representation formalism suitable for realistic applications.

The ARC-TEC project [Bernardi *et al.*, 1991] at the DFKI has considered a production planning problem in the field of mechanical engineering as a testbed for AI methodologies. Although it would be possible to present the main results of this thesis without any reference to this application, it is used for illustration purposes, since the representation and reasoning demands of this application were an important driving force for the research. In particular, the experiences made during the development and use of the hybrid compilation laboratory CoLab [Boley *et al.*, 1993] in the ARC-TEC project were valuable for the progress of this thesis.

CoLab comprises declarative components with pragmatic interfaces: a terminological formalism, a constraint system, a forward chaining system and a backward-chaining system. One challenge for the development of CoLab (and also for the thesis) was the requirement that the inference scheme underlying the production planning application can be realized. Chapter 2 gives a short introduction to the application domain.

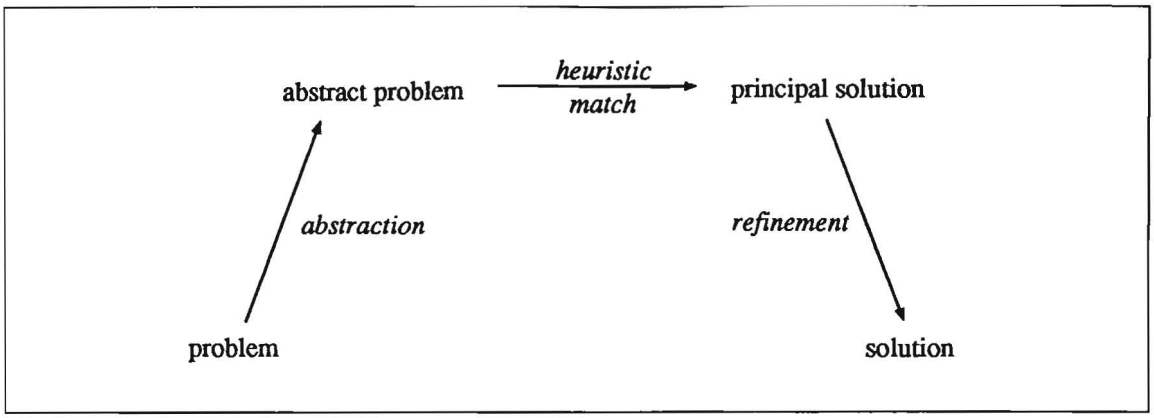

Figure 1.1: The Heuristic Classification Inference Scheme

1.1 Heuristic Classification

To demonstrate the results attained in the thesis, the next step is to describe the inference scheme underlying the application on a more abstract level.

It is well known that experience and heuristics are essential for problem solving in realistic domains. Clancey's *heuristic classification* is a simple inference scheme on the knowledge level ([Clancey, 1985], Figure 1.1) reflecting this general observation. The scheme comprises three main phases: an abstraction phase, a heuristic match, and a refinement phase. Expert systems of various task categories, such as diagnosis, configuration, and planning have been based on heuristic classification. In the ARC-TEC project a variation of Clancey's scheme has been identified as the model of expertise in the production planning application [Schmalhofer *et al.*, 1991].

The *abstraction phase* starts with concrete data (knowledge) of a problem and generates an abstract view of the data that contains *triggers* [Clancey, 1985], (also called 'application features' [Klauck *et al.*, 1991]) which are relevant for problem solving. Contrarily, the *refinement phase* collects, combines and instantiates principal (partial) solutions into a concrete solution. The *heuristic match* associates triggers in an abstract problem description (the output of abstraction) to principal (partial) solutions (the input of refinement). These heuristic associations between two different terminologies represent experimental experience and avoid reasoning from first principles, which often would cause severe performance problems.

1.2 The Terminological Approach

From a more general point of view the problem description produced by abstraction is mapped by a heuristic, non-deterministic match to descriptions of partial, principal solutions. Hence, description logics play an important role in this inference scheme.

Terminological formalisms in the tradition of KL-ONE [Brachman and Schmolze, 1985], also referred to as description logics, concept languages etc., can be used to

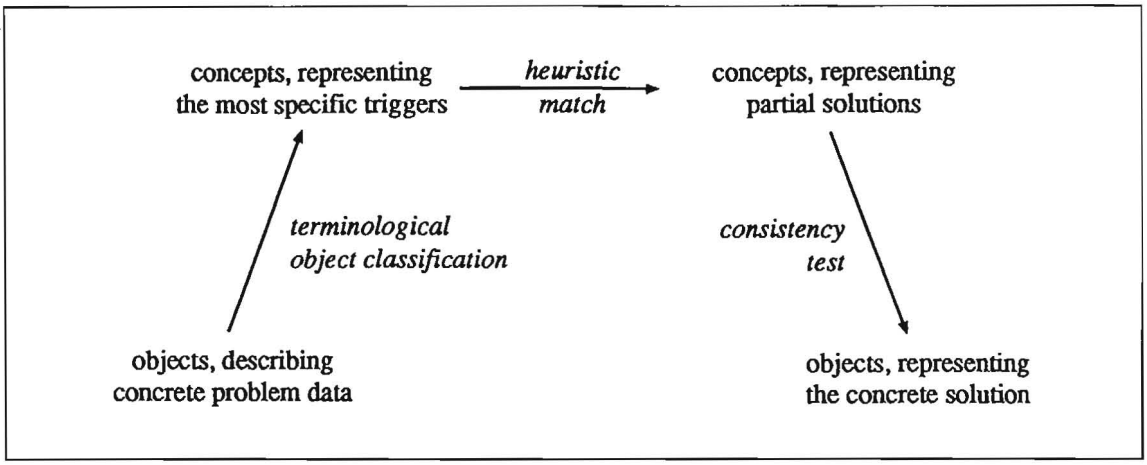

Figure 1.2: The Inference Scheme with Terminological Formalisms

represent the terminological knowledge of a particular problem domain on a formal basis. A terminological box (T-box) is used to represent concepts which can then be applied to objects in an assertional box (A-box). The formalisms provide reasoning services such as classification of objects and concepts with respect to a generalization hierarchy of concepts (Section 3).

At first glance the scheme can be implemented in a terminological formalism as depicted in Figure 1.2. The initial problem could be represented by objects in the A-box. The classification of objects with respect to a terminology of triggers would reveal the most specific triggers which apply to this object. Then a heuristic match, which could be implemented by trigger rules (e.g., [Brachman *et al.*, 1991]), would associate concepts of a terminology of partial, abstract solutions to the triggers. Finally, a consistency test, which expands the concept definitions involved, could check the feasibility of the solution.

Terminological reasoning does not have an intuitive operational semantics. Consequently, it should not be the responsibility of the knowledge engineer to check whether all intended queries will terminate in finite time. But these formalisms do have an intuitive, precise model-theoretic semantics. Therefore, it has been a research goal in the field of terminological knowledge representation to find the most expressive formalisms with decidable reasoning problems, or even better, with tractable inference algorithms (see e.g. [Levesque and Brachman, 1987; Donini *et al.*, 1991]). However the required decidability (tractability) imposes significant limitations on the expressiveness of these formalisms.

Section 2 exhibits some representation demands that occur in the above-mentioned mechanical engineering application and various other domains.

- *Role Interaction and Concrete Domains*
  It is often the case that properties or roles of an object are not independent. For example, a mother is older than her children, or the color of the hair of a good model should harmonize with the color of the clothes she is going to present.

The relation of the properties and roles in the first example can be described in terms of a concrete domain: the predicate $<$ over integers. In the second example the interaction is more abstract. Extensions of concept languages for both kinds of interaction are discussed in some detail in Chapter 3.

- *Sequences*
  A sequence is composed in a regular manner of a finite, bounded, but previously unknown number of smaller entities.

In Section 4.2 some principal limitations of terminological formalisms are discussed. It is recalled from [Baader and Hanschke, 1992] that a concept language that is capable of representing sequences of finite but unbounded size and that provides concrete domains has an undecidable subsumption problem. Other problems are related to the "part-of" relation and derived attributes which turn out to be relevant for abstraction [Hanschke and Hinkelmann, 1992] as well as refinement.

1.3 The Integrated Approach

How to overcome the limitations of terminological formalisms? One possibility is to combine the advantages of a concept language with respect to terminological knowledge representation and reasoning with the reasoning power of a more expressive but semidecidable formalism in a declarative manner.

Constraint Logic Programming (CLP) systems [van Hentenryck, 1989] provide goal-directed, top-down inferences and enhance search performance by means of constraint solving and propagation. Hence, these systems can be relevant for refinement. But, in general, they do not support terminological reasoning. LOGIN [Aït-Kaci and Nasr, 1986], which can be seen as a CLP system, integrates a fixed taxonomy in its feature-unification algorithm, which may be considered as a weak form of terminological reasoning. LIFE [Aït-Kaci and Podelski, 1991] is an extension of LOGIN that additionally provides functions. Since a function application is evaluated immediately, this may be considered as a data-driven, bottom-up computation. But this is very different from data-driven reasoning with multiple-premise production rules. Similarly, Oz [Würtz *et al.*, 1993], is a constraint programming language integrating goal-directed reasoning with constraint propagation, but it does not directly include data-driven inferences with rules of multiple premises, although it may be possible to implement this feature using the communication primitives **get** and **put**. The operational semantics of Oz is sound but not complete with respect to classical logic.

Production rule systems as well as bottom-up parsers and bottom-up logic programming systems are suitable for data-driven abstraction. But, again, these systems are not essentially hybrid and integrate only weak forms of terminological reasoning, if any.

Thus, there exist formalisms supporting terminological knowledge which is relevant for all phases, and expressive formalisms supporting abstraction and refinement, separately. There also exist (partial) pragmatic integrations of terminological or taxonomic knowledge representation with data-driven, goal-directed and constraint-based reasoning. MacGregor's LOOM [MacGregor, 1988] and the above-mentioned CoLab system are examples of pragmatic extensions integrating a terminological component. See [Firebaugh, 1988] for an overview of (commercially available) hybrid expert system shells with taxonomic components.

However, to combine a rule formalism (such as definite definitions in Horn logic) with an expressive condition logic (such as a terminological logic) in a declarative manner is a non-trivial research problem. For example, the operational semantics of a rule formalism usually does not capture the contra-position of rules.

For top-down reasoning the generalized CLP scheme proposed by Höhfeld and Smolka [Höhfeld and Smolka, 1988] is an interesting approach. It shows in a generic manner how constraint formalisms, which have to satisfy some weak requirements, can be combined with definite relations such that the operational semantics is sound and complete with respect to classical logic. This has been achieved by restricting the head of a rule to positive relational atoms and by considering conditional answers instead of taking the theorem-proving point of view.[1]

In fact in [Abecker and Hanschke, 1993] this scheme was employed to integrate terminological reasoning, goal-directed reasoning, and constraint based reasoning. Data-driven inferences, however, which are of particular interest for the abstraction phase, are not supported by this CLP formalism. See [Frühwirth and Hanschke, 1993] for an implementation approach of this formalism on top of Prolog and some examples from a simple configuration domain.

In [Hanschke and Hinkelmann, 1992] we combined terminological with data-driven rule-based reasoning for abstraction processes. The semantics of this production rule-like formalism is specified by a fixpoint operator based on 'constructive implication', i.e., a rule may only be applied to objects explicitly named in the current fact base.

In Chapter 5 a scheme is introduced that combines CLP and production rule-like inferences. The scheme takes a condition logic and constructs a rule formalism with rules of the form

$$\phi_0 \rightsquigarrow \phi_1 | \ldots | \phi_n$$

where the ϕ_i are formulas of the condition logic and $\rightsquigarrow$ is a kind of procedural implication, which is explained in more detail in Chapter 5. Informally, such a rule *says "if ϕ_0 is believed, then one of ϕ_1, ϕ_2, ...,ϕ_n is believed."*

The operational semantics of the rule formalism generalizes the way production rules (and trigger rules) are applied to a fact base. If a rule is triggered, one of the ϕ_i in the head is non-deterministically selected and added to the fact base. If an inconsistency occurs, backtracking takes place and another alternative is selected.

[1] See [Bürckert, 1991] for a combination of a constraint formalism with a resolution theorem prover.

For $n = 0$ the rule is a *denial* saying that whenever ϕ_0 is believed, the current state is inconsistent. For $n = 1$ the rules are very close to production rules. If ϕ_0 is very simple and $n > 1$, the operational semantics of these rules has much in common with SLD resolution (cf. Section 5.5.1).

For simple trigger rules $A \rightsquigarrow B$, A and B concepts, a semantics based on the epistemic operator K was proposed in [Donini *et al.*, 1992] that coincides with the common operational semantics: "If there is a 'known object' a in the fact base that is an A, then add $B(a)$ to the fact base." Surprisingly, this semantics does not carry over to the rule formalism considered here.

To get a reasonable model-theoretic semantics it is necessary to refine this epistemic logic (which, by the way, is similar to the logics considered in [Levesque, 1984; Reiter, 1990; Lifschitz, 1991]). The formalization of a "really" minimal belief logic in Section 5.2 resolves certain problems (Section 5.1.2) related to (*i*) the absence of a unique-name assumption, (*ii*) certain interactions of *more than one* existential quantifications with an belief operator, and (*iii*) formalisms that are expressive enough to restrict in a formula ϕ the cardinality of the domain of the models of ϕ. The operational semantics is sound and complete with respect to this epistemic logic.

Consequently, the rule scheme is a hybrid, generic declarative formalism integrating deterministic, data-driven, bottom-up reasoning (as required for abstraction) with non-deterministic, goal-directed, top-down search (as required by the association and the refinement phase). If the scheme is applied to a terminological formalism that is extended by concrete domains (Section 5.6), this results in a declarative integration of terminological, constraint-based, data-driven and goal-directed reasoning. Chapter 5 concludes by showing how the representation and reasoning problems that could not be handled by the terminological formalisms (Section 4.2) can be handled in the integrated formalism (Section 5.6.2).

The intended reader of this thesis should be familiar with the basics of formal logic, logic programming and symbolic knowledge representation. It is not required in any way that she or he has a background in mechanical engineering.

Chapter 2

An Application Domain

The application domain that has been investigated in the ARC-TEC project is production planning for CNC lathes. More precisely, the work has been motivated by the following scenario:

> Given the geometry of a rotationally-symmetric workpiece, generate the process plans as abstract NC macros for turning the workpiece on a CNC lathe.

Reasoning in this application follows a scheme (Figure 2.1) that is inspired by Clancey's *heuristic classification* [Schmalhofer *et al.*, 1991; Bernardi *et al.*, 1992b]: The input is a CAD drawing describing the workpiece in terms of primitive surfaces and basic technological data. The *abstraction* phase generates a schematic description of the workpiece in terms of *(CAD/CAM) features* [Klauck *et al.*, 1991; Bernardi *et al.*, 1992a]. Such features (which are the triggers in Clancey's scheme) are often associated with parts of the workpiece that are characteristic with respect to how these parts (or the whole lathe) may be manufactured. The second phase heuristically

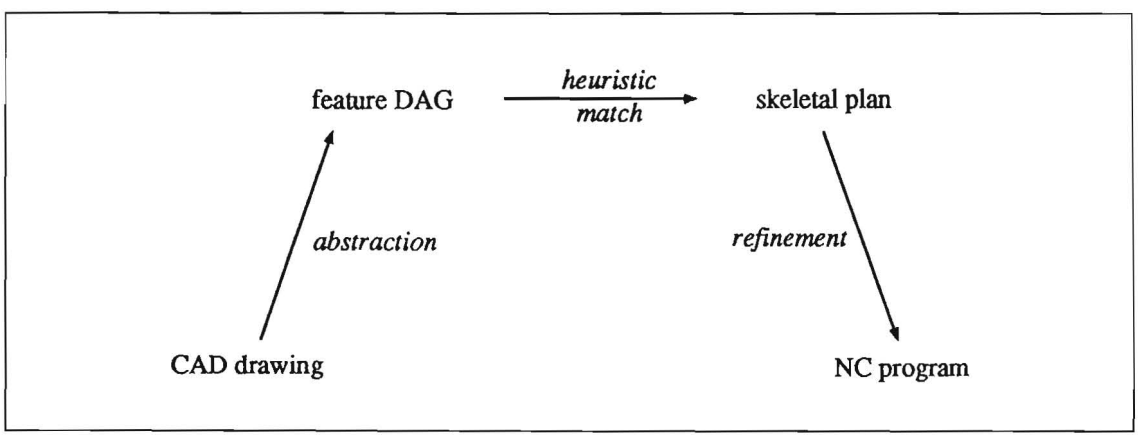

Figure 2.1: Heuristic Classification Applied to Production Planning of CNC Lathes

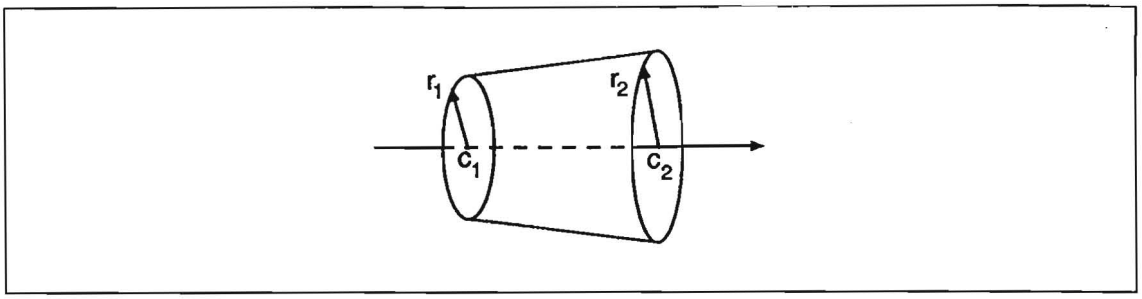

Figure 2.2: A Truncated Cone

matches *skeletal (production) plans* to the features. Finally, the third phase *refines* and merges the skeletal plans to a complete program for CNC machines.

This problem domain requires, among other things, the representation of geometric primitives of the workpiece taken from the CAD drawing, as well as other relevant technological data. The features characterizing the workpiece and the skeletal plans are also important parts of the knowledge necessary for solving the application problem. If all this could be expressed in a concept language, the inference scheme could be mapped naturally into a terminological framework:

- Arrange the features represented as concepts in a generalization hierarchy using the concept-classification service.
- Represent a particular CAD drawing of the workpiece with its geometric and technological information as instances of appropriate concepts in the A-box.
- Employ the object-classification service [Nebel, 1990] to compute the most specific concepts that apply to the particular lathe.
- Associate the skeletal plans to the production features detected by means of trigger rules (cf. Chapter 5).
- Check the feasibility of the solution with the consistency test for A-boxes.

However, it is easy to see that conventional concept languages cannot be used for adequately representing this problem domain. Consider for example the concept of a truncated cone (see Figure 2.2). Since in this domain geometric objects are regarded as being fixed to an axis, a truncated cone can be characterized by four real numbers, two for its radii and two for the corresponding centers. But of course, not each quadruple of real numbers represents a truncated cone. Hence, the values have to be restricted such that the radii are positive and the surface of the truncated cone does not degenerate to a line, a circle, or even a point. It seems to be impossible to represent these restrictions using only "abstract" concept terms without reference to predicates over, for example, real numbers. This requirement reveals the need for an integration of *concrete domains*.

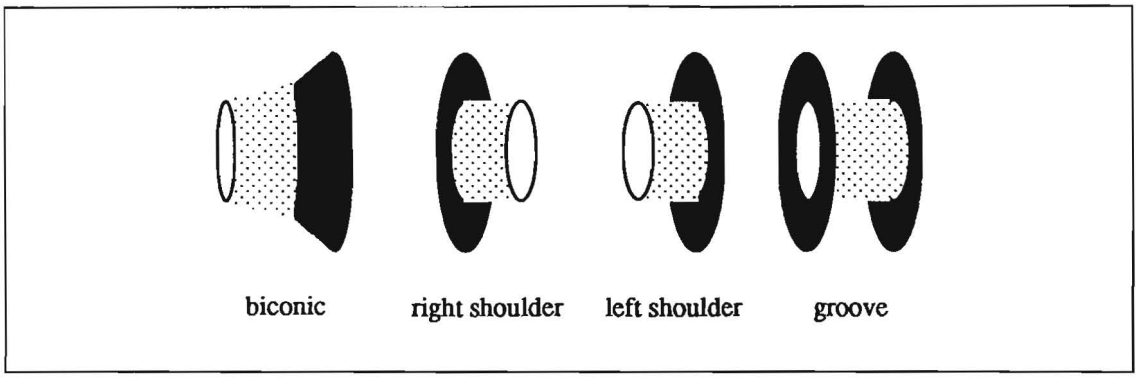

Figure 2.3: Some Simple Features

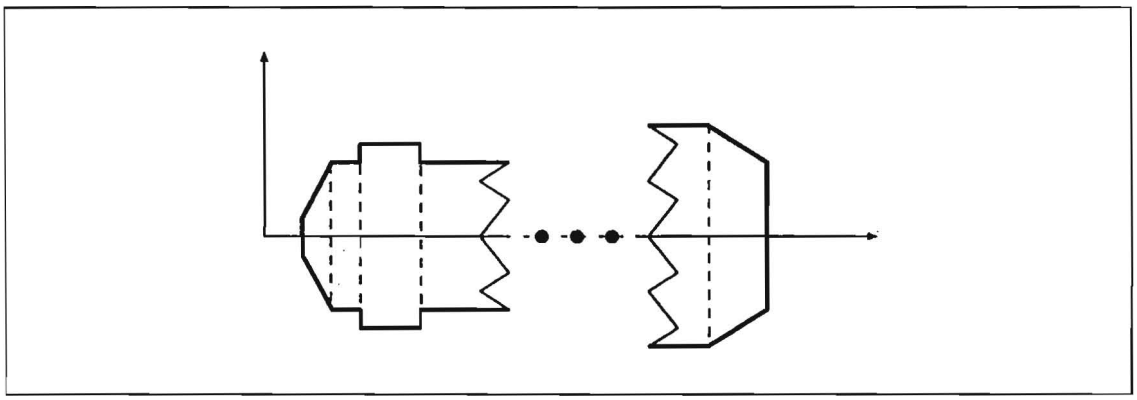

Figure 2.4: The Varying-Size Aspect of a Lathe

Features often correspond to some neighbored truncated cones. For example a 'biconic' just consists of two neighboring truncated cones (Figure 2.3). A 'left shoulder' (resp. 'right shoulder') is a 'biconic' with a surface line approximating the shape of a human shoulder. A 'groove' is a combination of a left and a right shoulder that share a common ground.

Note that it is essential for a biconic that its truncated cones are neighbored. Similarly, the definition of a groove relys on a relation between its subcomponents, also. From an abstract point of view, the adequate representation of a truncated cone, a biconic, or a groove requires the formalization of interrelations of parameters or components. Since terminological formalisms relate parameters or components to an object via roles, this issue has been termed *role interaction*. The following chapter investigates extensions of concept languages for role interaction.

Other features such as an 'ascending sequence of truncated cones' correspond to sequences of simpler features or geometric primitives with some common property and certain interrelations. These sequences have a finite, but varying and not a priori bounded length (Figure 2.4).

Chapter 3

Terminological Knowledge Representation

In the previous chapter role interaction has been identified as an important representation and reasoning demand. This chapter investigates extensions of concept languages dealing with this issue and introduces a new reasoning service, called *A-box subsumption.* In the context of the rule scheme (Chapter 5) this service will be used to check whether a 'fact base' (i.e., A-box) entails a premise of a rule.

3.1 Introduction

Concept languages based on KL-ONE [Brachman and Schmolze, 1985] are mostly used to represent the terminological knowledge of a particular problem domain on an abstract logical level. To describe this kind of knowledge, one starts with atomic concepts and roles, and defines new concepts using the operations provided by the language. Concepts can be considered as unary predicates which are interpreted as sets of individuals, and roles as binary predicates which are interpreted as binary relations between individuals. Examples for atomic concepts may be human and female, and for roles friend and enemy. Many terminological formalisms concentrate on the following three categories of operators to build a terminology:

- *Boolean connectives* ($\sqcap$, $\sqcup$, and $\neg$) that allow concepts to be combined without any direct reference to their internal structure. For example, if the logical connective conjunction is present as a language construct, one may describe the concept woman as "humans who are female", and represent it by the expression human $\sqcap$ female.

- *Role-forming operators* that allow new roles to be defined. For example the composition ($\circ$) allows the role "friend of enemy" to be represented by enemy$\circ$ friend.

- Operators on *role fillers* that allow the 'internal' structure of the concepts to be operated on. Many languages provide quantification over role fillers which allows, for example, the concept "human with a friend" (resp. "human with only female friends") to be described by the expression human $\sqcap \exists$friend.human (resp. human$\sqcap\forall$friend.female). An interesting subclass of operators on role fillers are the operators for *role interaction.* The frequently used *number restrictions* can be seen as a degenerated form to specify role interaction (on one role). For example, the concept lucky-human could be defined as $\exists^{>100}$friend $\sqcap$ $\exists^{<2}$enemy. As soon as an individual belonging to this concept has two role fillers for enemy, it can be deduced that they are equal.

The kind of models that can be specified by the operators considered so far is quite restricted. If a concept C is satisfiable, then it is satisfiable by an interpretation that arranges its individuals in a tree structure (cf. Figure 3.1). For example, it is possible to require that the members of a concept have role fillers for a role R, say an individual a, and a role S, say b. But it is not possible to specify that a equals b or that a and b have any common (transitive) role-filler, or that their respective role-fillers are in any relation to each other.

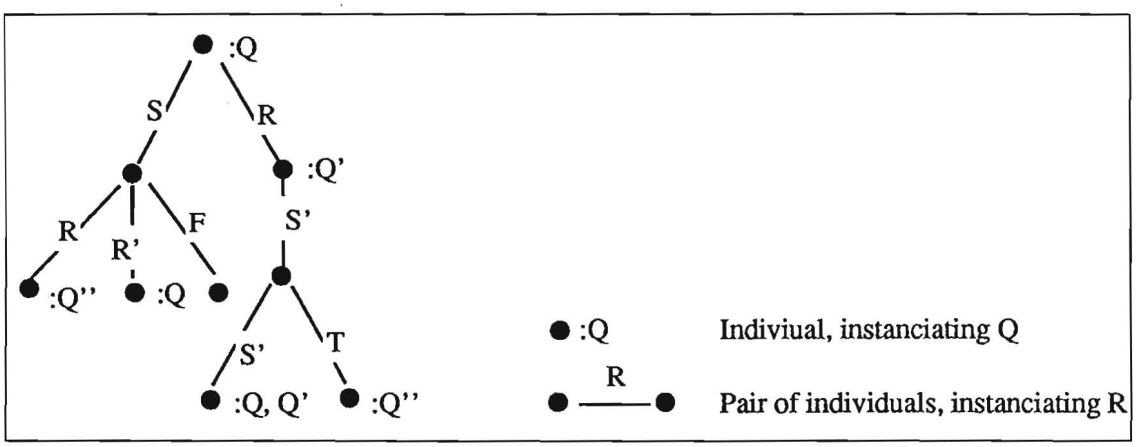

Figure 3.1: A Typical Model Structure of a Conventional Concept Term

So there is a need for additional means to specify role interaction. The classical prototypes of this kind of operators are the *structural descriptions* and *role-value maps* (RVMs; see Section 3.3 for a definition) that are discussed and motivated, for example, in [Brachman and Schmolze, 1985].

An RVM would allow one to specify that the set of all friends of an individual is equal to the set of all enemies (which may be true for some people if one looks at some never ending soap operas): enemy $=_{\text{RVM}}$ friend where enemy and friend are roles. A typical model structure is depicted in Figure 3.2.

In [Schmidt-Schauß, 1989; Patel-Schneider, 1989] it has been shown that a concept language with RVMs and a few other common operators has an undecidable subsumption problem. As a reaction on this disappointing negative result, RVMs have

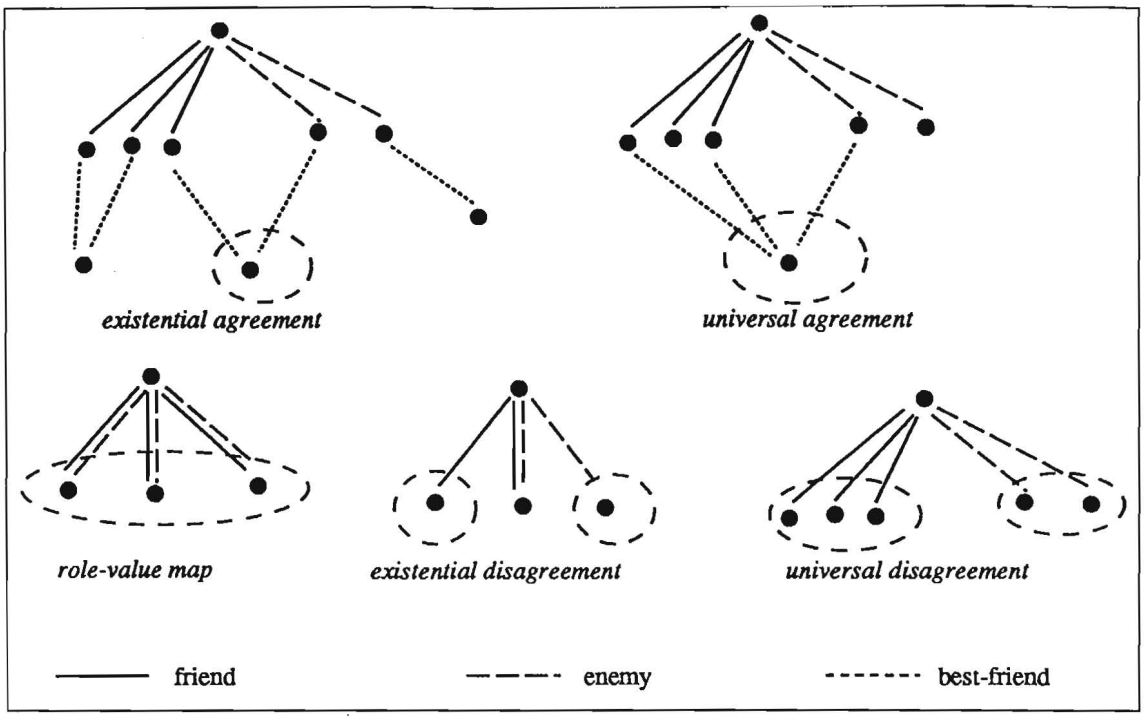

Figure 3.2: Typical Model Structures with Role/Attribute Interaction

been restricted in existing systems to attribute agreements, see for example [Borgida *et al.*, 1989]. *Attributes* are functional roles and are sometimes also called features. I.e., they have at most one role filler per object. Let best-friend and main-enemy be attributes. Then an individual belongs to the concept main-enemy $=_{\text{RVM}}$ best-friend if it does not have a main enemy, or if it does not have a best friend, or if its best friend is at the same time its main enemy.[1]

In this chapter several other operators for specifying interaction of role and attribute fillers are investigated. The *existential role/attribute agreement* can be used to specify that there is at least one enemy that is also a friend: $\exists$(enemy = friend). If this operator is restricted to attribute chainings it is just the same-as operator in CLASSIC.

The expression $\exists$(enemy $\circ$ best-friend = friend $\circ$ best-friend) represents that there is at least one enemy and one friend who have the same best-friend. The *universal* agreement is used in the expression $\forall$(enemy $\circ$ best-friend = friend $\circ$ best-friend) to formalize that the best-friends of all friends and enemies are the same (cf. Figure 3.2). On attribute chainings this construct agrees with the RVMs.

The *existential role/attribute disagreement* can express that there is at least one enemy and one friend that are not identical: $\exists$(enemy $\neq$ friend). The expression $\forall$(enemy $\neq$ friend) says that each member has only true friends and true enemies—

[1] Actually, in CLASSIC the same-as operator requires the existence of one main-enemy and one best-friend.

there is no filler that is both a friend and an enemy (cf. Figure 3.2).

Although it is at least not obvious how RVMs (on roles) can be simulated by this group of operators, it turns out that the existential and universal agreements lead to an undecidable subsumption problem (Section 3.3), also.

Section 3.4 introduces a new concept language which is able to relate fillers of role/attribute chainings. The main idea is to replace the general "=" (resp. "$\neq$") above, by abstract, not further defined predicates or by predicates of a *concrete domain.* In [Baader and Hanschke, 1991a] we already proposed an extension scheme with concrete domains, but there, the predicates are only applied to chainings of attributes. Following [Hanschke, 1992] this chapter generalizes this extension scheme considerably.

As an example, consider the classic (toy) domain of families. Let age, wife, and husband be attributes, child a role, and male, human not further defined concepts. Then the concept of a family could be represented by

$$\begin{aligned} \mathsf{woman} &= \mathsf{human} \sqcap \mathsf{female} \\ \mathsf{man} &= \mathsf{human} \sqcap \neg\mathsf{female} \\ \mathsf{family} &= \exists \mathsf{husband.man} \sqcap \exists \mathsf{wife.woman} \sqcap \forall \mathsf{child.human} \end{aligned}$$

The specification can be further refined by enforcing that there is a marriage certificate and that children are younger than their parents.

$$\begin{aligned} \mathsf{normal\text{-}family} = \mathsf{family} \sqcap \\ \forall(\mathsf{child} \circ \mathsf{age} < \mathsf{husband} \circ \mathsf{age}) \sqcap \\ \forall(\mathsf{child} \circ \mathsf{age} < \mathsf{wife} \circ \mathsf{age}) \sqcap \\ \forall \mathsf{husband, wife.marriage\text{-}certificate} \end{aligned}$$

Here the concrete predicate "<" and an abstract binary predicate marriage-certificate are used to formulate the additional requirements.

With concrete predicates concept definitions can be 'grounded'. This is, for example, useful in technical domains where one is confronted with coordinates in space (predicates over rational numbers) or technical data sheets or tables (extensionally defined predicates).

The formalism considered so far deals primarily with intensional concept definitions and is referred to as the *terminological box (T-box).* The *assertional box (A-box)* is a formalism to make assertions about instances of the roles, attributes, concepts and predicates introduced in the T-box. The A-box formalism, usually provides services such as consistency test, membership test, and object classification.

In the rule formalisms considered in Chapter 5 A-boxes [2] will occur in premises. In order to check whether a global A-box entails an instance of such a premise a service is needed that compares A-boxes with respect to generality. This new service is called *A-box subsumption.* It can also be used to compare rules with terminological premises [Hanschke and Meyer, 1992]. The corresponding reasoning problem is undecidable

[2] By abuse of notation a collection of assertions is also called A-box.

for general A-boxes. The notion of *rooted* A-boxes defined in Section 3.5 leads to a restriction of the A-box subsumption problem that is decidable and still useful.

Another way to understand A-box subsumption (and its name) is as follows: Let the objects in an A-box $\mathcal{A}$ be split into two disjoint sets $\{x_1, \cdots, x_n\}$, $n > 0$, and Y. If the objects $y \in Y$ are considered as being existentially quantified, $\mathcal{A}$ induces an n-ary predicate $\lambda x_1, \cdots, x_n.\mathcal{A}$, which is regarded as an "n-ary concept". A-box subsumption is then the subsumption service in this extended concept language. Note that this "concept formalisms" uses objects in its concept definitions.

3.2 The Basic Language

This section introduces the language ALCF as a prototypical conventional concept language. It will be the starting point for the extensions described in the following sections.

Definition 3.2.1 (T-box syntax) Concept terms *are built from concept, role, and attribute names using concept-forming operators. If C and D are syntactic variables for concept terms and R is a role or attribute name, then*

$C \sqcap D$ (conjunction),
$C \sqcup D$ (disjunction),
$\neg C$ (negation)
$\exists R.C$ (exists-in restriction), *and*
$\forall R.C$ (value restriction)

are concept terms.

Let A be a concept name and let D be a concept term. Then $A = D$ is a terminological axiom. *A* terminology (T-box) *is a finite set $\mathcal{T}$ of terminological axioms with the additional restrictions that no concept name appears more than once as a left hand side of a definition, and $\mathcal{T}$ contains no cyclic definitions.*[3]

A concept name that does not occur on the left side of a concept definition is called primitive. □

Please note that the exists-in and the value restrictions are not only defined for roles but also for attributes. The next definition gives a model-theoretic semantics for the language introduced in Definition 3.2.1.

Definition 3.2.2 (T-box semantics) *An* interpretation $\mathcal{I}$ *for* ALCF *consists of a set* $\text{DOM}_\mathcal{I}$ *and an interpretation function. The interpretation function associates with each concept name A a subset $A^\mathcal{I}$ of* $\text{DOM}_\mathcal{I}$*, with each role name R a binary relation $R^\mathcal{I}$ on* $\text{DOM}_\mathcal{I}$*, i.e., a subset of* $\text{DOM}_\mathcal{I} \times \text{DOM}_\mathcal{I}$*, and with each attribute name f a partial function $f^\mathcal{I}$ from* $\text{DOM}_\mathcal{I}$ *into* $\text{DOM}_\mathcal{I}$*. For such a partial function $f^\mathcal{I}$ the expression $f^\mathcal{I}(x) = y$ is sometimes written as $(x, y) \in f^\mathcal{I}$.*

[3] See [Nebel, 1990; Baader, 1990] for a treatment of cyclic definitions in concept languages.

The interpretation function—which gives an interpretation for atomic terms—can be extended to arbitrary concept terms as follows: Let C and D be concept terms and let R be a role or attribute name. Assume that $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$ are already defined. Then

1. $a \in (C \sqcup D)^{\mathcal{I}}$ *iff* $a \in C^{\mathcal{I}}$ *or* $a \in D^{\mathcal{I}}$,
   $a \in (C \sqcap D)^{\mathcal{I}}$ *iff* $a \in C^{\mathcal{I}}$ *and* $a \in D^{\mathcal{I}}$,
   $a \in (\neg C)^{\mathcal{I}}$ *iff* $a \in \mathrm{DOM}_{\mathcal{I}} \setminus C^{\mathcal{I}}$,

2. $a \in (\forall R.C)^{\mathcal{I}}$ *iff*
   for all b *with* $(a,b) \in R^{\mathcal{I}}$ *we have* $b \in C^{\mathcal{I}}$, *and*
   $a \in (\exists R.C)^{\mathcal{I}}$ *iff*
   there exists b *with* $(a,b) \in R^{\mathcal{I}}$ *and* $b \in C^{\mathcal{I}}$.

An interpretation $\mathcal{I}$ is a model *of the T-box $\mathcal{T}$ iff it satisfies $A^{\mathcal{I}} = D^{\mathcal{I}}$ for all terminological axioms $A = D$ in $\mathcal{T}$.* □

An important service terminological representation systems provide is computing the subsumption hierarchy, i.e., computing the subconcept-superconcept relationships between the concepts of a T-box. This inferential service is usually called *concept classification.* The model-theoretic semantics introduced above allows the following formal definition of subsumption and satisfiability.

Definition 3.2.3 (T-box services) *Let $\mathcal{T}$ be a T-box and let C, D be concepts. Then D* subsumes *C with respect to $\mathcal{T}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$. A concept C is* satisfiable *if there is a model $\mathcal{I}$ of $\mathcal{T}$ that satisfies C, i.e., $C^{\mathcal{I}}$ is not empty.* □

Note that D *subsumes* C iff $C \sqcap \neg D$ is satisfiable and that C is satisfiable if $\bot$ subsumes C where $\bot$ stands for the empty concept that could be defined by $\bot = D \sqcap \neg D$, for some concept D.

All extensions of ALCF in the present chapter involve attribute/role chainings, which are built from role and attribute names with the binary, associative infix operator $\circ$ which is interpreted according to

$(a,b) \in (R_1 \circ R_2)^{\mathcal{I}}$ iff
there is a c with $(a,c) \in R_1^{\mathcal{I}}$ and $(c,b) \in R_2^{\mathcal{I}}$

The special attribute name ϵ is always interpreted as identity.

3.3 Equality Based Operators

In this section a concept language based on ALCF with additional concept forming operators, called existential and universal role/attribute (dis)agreements, is formally defined. These concept forming operators are based on equality and negated equality.

Let $u =_{\text{RVM}} v$ be the original RVM construct, where u and v are two, possibly empty, chainings of roles and attributes. An individual a belongs to the concept $u =_{\text{RVM}} v$ iff the two sets of (transitive) role-fillers of u and v are identical. Formally, an interpretation extends to the RVMs according to:[4]

$$a \in (u =_{\text{RVM}} v)^{\mathcal{I}} \quad \text{iff} \quad au^{\mathcal{I}} = av^{\mathcal{I}}$$

Note that each of the following constructs is different from the RVM construct.

Definition 3.3.1 (equality-based operators) *Let u and v be two role chainings. Then the syntax of the concept forming operators based on equality and negated equality is defined as follows:*

$\forall(u = v)$ (universal agreement)
$\forall(u \neq v)$ (universal disagreement)
$\exists(u = v)$ (existential agreement)
$\exists(u \neq v)$ (existential disagreement)

To define the semantics of these new operators the interpretation function is extended as follows:

$a \in \forall(u = v)^{\mathcal{I}}$ *iff*
for all b, c with $(a, b) \in v^{\mathcal{I}}$ and $(a, c) \in u^{\mathcal{I}}$ we have $b = c$

$a \in \forall(u \neq v)^{\mathcal{I}}$ *iff*
for all b, c with $(a, b) \in v^{\mathcal{I}}$ and $(a, c) \in u^{\mathcal{I}}$ we have $b \neq c$

$a \in \exists(u = v)^{\mathcal{I}}$ *iff*
there exists b with $(a, b) \in v^{\mathcal{I}}$ and $(a, b) \in u^{\mathcal{I}}$

$a \in \exists(u \neq v)^{\mathcal{I}}$ *iff*
there exist b, c with $(a, b) \in v^{\mathcal{I}}$ and $(a, c) \in u^{\mathcal{I}}$ and $b \neq c$ □

If u and v are attribute agreements, $u =_{\text{RVM}} v$ and $\forall(u = v)$ are equivalent concepts. This is not the case if u and/or v contains a role. Moreover, it is at least not obvious how RVMs with roles can be simulated by the equality-based operators. Unfortunately, ALCF together with the constructs of the previous definition does not have a decidable subsumption problem, either.

This will be shown by a reduction of the *word problem for semi-groups* to the subsumption problem in the concept language. First, the definition of the word problem is recalled. Let Σ be a finite alphabet, let Σ^* be the set of finite, possibly empty words over Σ, and let ϵ be the empty word. Then a set $S = \{l_i = r_i |\ l_i, r_i \in \Sigma^*, i = 1, \ldots, m\}$ is called a *finite presentation of a semi-group*. This set induces a binary relation $\rightarrow_S$ on Σ^*:

[4]For a binary relation r and an object a the expression ar is defined as the set $\{b |\ r(a, b)\}$.

$u \rightarrow_S v$ iff

there are words w_1, $w_2 \in \Sigma^*$, and an $l = r \in S$ such that $u = w_1 l w_2$ and $v = w_1 r w_2$.

By $\sim_S$ the reflexive, transitive, and symmetric closure of $\rightarrow_S$ is denoted. It is well known that a finite presentation S exists consisting of seven equations over a two-element alphabet, $\Sigma = \{a, b\}$ say, such that it is undecidable for two words u and v whether $u \sim_S v$ holds or not (see, for instance [Boone, 1959]).

Now let this system S be given. For the two elements a, $b \in \Sigma$ two attributes a, b are introduced, respectively. Let start, left, right be additional attributes, let back, forth be additional role names. Then for two words u, $v \in \Sigma^*$ the following concept definition scheme is introduced:

$$\begin{aligned} \mathsf{eq}_{u,v} = \; & \exists(\mathsf{left} = \mathsf{start} \circ u) \sqcap \\ & \exists(\mathsf{right} = \mathsf{start} \circ v) \sqcap \\ & \exists(\mathsf{forth} = \mathsf{start}) \end{aligned}$$

Let $u = f_1 \cdots f_m$ and $v = g_1 \cdots g_n$. Then for any model $\mathcal{I}$ (of the terminology up to this point) satisfying $\mathsf{eq}_{u,v}$ there are (not necessarily distinct) objects, c, a_0, $a_1, \cdots, a_m$, b_0, $b_1, \cdots, b_n$ such that

1. $(a_{i-1}, a_i) \in f_i^{\mathcal{I}}$, for $0 < i \leq m$, and $(b_{i-1}, b_i) \in g_i^{\mathcal{I}}$, for $0 < i \leq n$
2. $a_0 = b_0$, $(c, a_0) \in \mathsf{start}$, $(c, a_0) \in \mathsf{forth}$, $(c, a_m) \in \mathsf{left}$, and $(c, b_n) \in \mathsf{right}$.

This attribute/role structure is depicted in Figure 3.3.

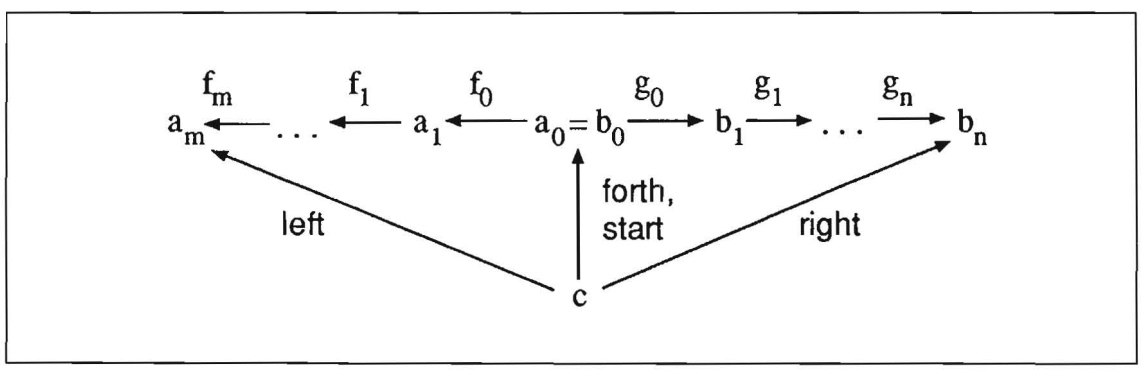

Figure 3.3: Representing $f_1 \cdots f_m = g_1 \cdots g_n$

A single equation $l = r \in S$ can be modeled by a concept $\mathsf{equation}_{l=r}$ defined by the following scheme:

$$\mathsf{equation}_{l=r} = \exists(l = r)$$

Assume that the model $\mathcal{I}$ satisfies $\mathsf{equation}_{l=r}$ and that $a_0 \in \mathsf{equation}_{l=r}^{\mathcal{I}}$. Then for $w \in \Sigma^*$ we have $(lw)^{\mathcal{I}}(a_0) = (rw)^{\mathcal{I}}(a_0)$. The presentation $S = \{e_1, \ldots, e_7\}$ can now be easily represented as

$$\mathsf{localS} = \mathsf{equation}_{e_1} \sqcap \ldots \sqcap \mathsf{equation}_{e_7}.$$

But how can this restriction be imposed on each element x for which there is a $w \in \Sigma^*$ such that $w^{\mathcal{I}}(a_0) = x$? Since the concept language does not provide transitive closure or cyclic definitions, the element c in Figure 3.3 is used as a 'relay that refreshes' the restriction. Consider, the following concept definition scheme:

$$\begin{aligned} \mathsf{loop}_\sigma = \; & \forall \mathsf{forth}.\forall\sigma.\exists(\mathsf{back} \circ \mathsf{forth} = \epsilon) \sqcap \\ & \forall(\mathsf{forth} \circ \sigma \circ \mathsf{back} = \epsilon) \end{aligned}$$

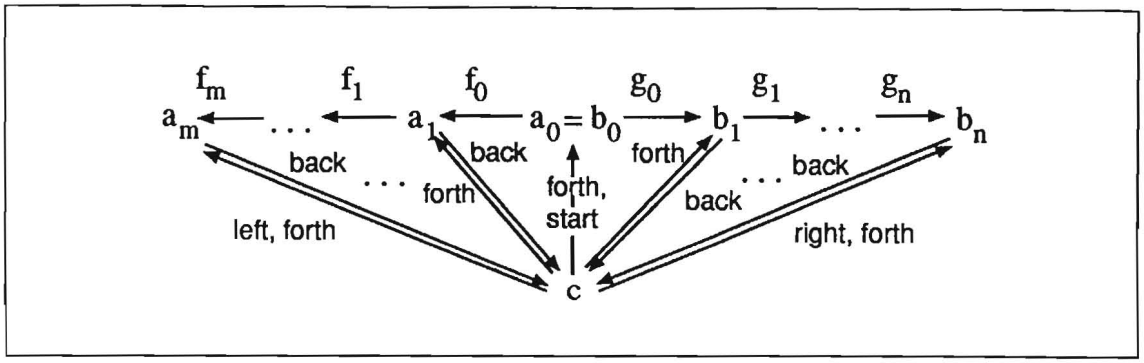

Figure 3.4: Repeating back and forth

Any model of the concept $\mathsf{eq}_{u,v} \sqcap \mathsf{loop}_\mathsf{a} \sqcap \mathsf{loop}_\mathsf{b}$ leads to a role/attribute structure similar to the structure depicted in Figure 3.4. More precisely, c has those x as role fillers for forth that can be reached from a_0 by a word $w \in \Sigma^*$. Now it is easy to impose the requirements of S on each of these elements: $\mathsf{globalS} = \forall\mathsf{forth}.\mathsf{localS}$

Proposition 3.3.2 *Given two words u, $v \in \Sigma^*$*

$$\begin{aligned} & \exists(\mathsf{left} = \mathsf{right}) \textit{ subsumes } \mathsf{eq}_{u,v} \sqcap \mathsf{loop}_\mathsf{a} \sqcap \mathsf{loop}_\mathsf{b} \sqcap \mathsf{globalS} \\ \textit{iff } & u \sim_S v. \end{aligned}$$

Proof.
1) *Assume that $u \sim_S v$:*
Let $\mathcal{I}$ be a model of the above concept definitions, and let c be in $(\mathsf{eq}_{u,v} \sqcap \mathsf{loop}_\mathsf{a} \sqcap \mathsf{loop}_\mathsf{b} \sqcap \mathsf{globalS})^{\mathcal{I}}$. Relying on the above construction the following can be proved:

> If $w^{\mathcal{I}}$ is defined on $\mathsf{start}^{\mathcal{I}}(\mathsf{c})$, and $w \rightarrow_S w'$ or $w' \rightarrow_S w$ then $w^{\mathcal{I}}(\mathsf{start}^{\mathcal{I}}(\mathsf{c})) = w'^{\mathcal{I}}(\mathsf{start}^{\mathcal{I}}(\mathsf{c}))$.

By definition there is a finite derivation of $u \sim_S v$ in terms of the symmetric closure of $\rightarrow_S$ and thus, $u^{\mathcal{I}}(\mathsf{start}^{\mathcal{I}}(\mathsf{c})) = v^{\mathcal{I}}(\mathsf{start}^{\mathcal{I}}(\mathsf{c}))$ and $\mathsf{left}^{\mathcal{I}}(c) = \mathsf{right}^{\mathcal{I}}(c)$. This implies $c \in \exists(\mathsf{left} = \mathsf{right})$.

2) *Assume that not $u \sim_S v$:*
The interpretation constructed below is a model of the above concept definitions and a counter example to the subsumption relation in question.

Let $\text{DOM}_{\mathcal{I}} = \Sigma^*/_{\sim_S} \cup \{c\}$ where $\Sigma^*/_{\sim_S}$ is the set of equivalence classes induced by the congruence relation $\sim_S$. The partial functions $\mathsf{a}^{\mathcal{I}}$ and $\mathsf{b}^{\mathcal{I}}$ are defined as left multiplications for all $[x] \in \Sigma^*/_{\sim_S}$:

$$\mathsf{a}^{\mathcal{I}}([x]) = [ax] \text{ and } \mathsf{b}^{\mathcal{I}}([x]) = [bx].$$

The other roles and attributes are defined as suggested by the construction:

1. $\mathsf{left}^{\mathcal{I}}(c) = [u]$, $\mathsf{right}^{\mathcal{I}}(c) = [v]$, and $\mathsf{start}^{\mathcal{I}}(c) = [\epsilon]$,
2. $(c, x) \in \mathsf{forth}^{\mathcal{I}}$, for every $x \in \Sigma^*/_{\sim_S}$, and
3. $(x, y) \in \mathsf{back}^{\mathcal{I}}$ if $(y, x) \in \mathsf{forth}^{\mathcal{I}}$. □

Corollary 3.3.3 *The subsumption problem in a concept language based on* ALCF *and extended by the* equality-based operators *universal and existential agreement is undecidable.* □

This result shows that, as long as equality is involved, it is wise to restrict oneselfs to attributes.

3.4 Operators with Predicates

The subsumption problem remains decidable if the equality-based operators are restricted to chainings of attributes. The reduction in the undecidability proof in the previous section relied heavily on the possibility to specify cyclic role structures (for instance, $\exists(\mathsf{back} \circ \mathsf{forth} = \epsilon)$).

In this section two ideas are developed that remove the capability to specify this kind of cyclic structure from the concept language. The first idea is to replace the equality in the equality-based operators by uninterpreted, possibly negated n-ary predicate symbols. The second idea is to split the interpretation domain into two separate domains: the *abstract* and the *concrete domain* [Baader and Hanschke, 1991a]. Role and attribute fillers can now be restricted by predicates of the concrete domain, also. But concepts are always subsets of the abstract domain.

Together with attribute (dis)agreements the abstract and the concrete predicate based operators are a powerful, still decidable, means to specify structural properties.

3.4.1 Concrete Domains

Before the concept forming operators are introduced the notion "concrete domain" has to be formalized.

Definition 3.4.1 *A* concrete domain $\mathcal{D}$ *consists of a set* $\text{DOM}_{\mathcal{D}}$*, the domain of* $\mathcal{D}$*, and a set* $\text{pred}(\mathcal{D})$*, the predicate names of* $\mathcal{D}$*. Each predicate name* p *is associated with an arity* n*, and an* n*-ary predicate* $p^{\mathcal{D}} \subseteq \text{DOM}_{\mathcal{D}}^n$. □

An important example is the concrete domain $\mathcal{R}$ of real arithmetic. The domain of $\mathcal{R}$ is the set of all real numbers, and the predicates of $\mathcal{R}$ are given by formulae which are built by first order means (i.e., by using logical connectives and quantifiers) from equalities and inequalities between integer polynomials in several indeterminates.[5] For example, $x + z^2 = y$ is an equality between the polynomials $p(x,z) = x + z^2$ and $q(y) = y$; and $x > y$ is an inequality between very simple polynomials. From these equalities and inequalities one can e.g. build the formulae $\exists z(x + z^2 = y)$ and $\exists z(x+z^2 = y) \vee (x > y)$. The first formula yields a predicate name of arity 2 (since it has two free variables), and it is easy to see that the associated predicate is $\{(r,s) |\ r$ and s are real numbers and $r \leq s\}$. Consequently, the predicate associated to the second formula is $\{(r,s);\ r$ and s are real numbers$\} = \text{DOM}_{\mathcal{R}} \times \text{DOM}_{\mathcal{R}}$.

Extensionally defined predicates as in finite-domain constraint systems or relational databases induce another important concrete domain. See [Steinle, 1993] for details.

To get inference algorithms for the extended concept language which will be introduced below, the concrete domain has to satisfy some additional properties.

For technical reasons the set of predicate names of the concrete domain is required to be *closed under negation*, e.g., if p is an n-ary predicate name in $pred(\mathcal{D})$ then a predicate name q in $pred(\mathcal{D})$ has to exist such that $q^{\mathcal{D}} = \text{DOM}_{\mathcal{D}}^n \setminus p^{\mathcal{D}}$. In addition, a unary predicate name is needed which denotes the predicate $\text{DOM}_{\mathcal{D}}$.

The property which will be formulated now clarifies what kind of reasoning mechanisms are required in the concrete domain. Let p_1, ..., p_k be k (not necessarily different) predicate names in $pred(\mathcal{D})$ of arities n_1, ..., n_k. Consider the conjunction

$$\bigwedge_{i=1}^{k} p_i(\underline{x}^{(i)}).$$

Here $\underline{x}^{(i)}$ stands for an n_i-tuple $(x_1^{(i)}, ..., x_{n_i}^{(i)})$ of variables. It is important to note that neither all variables in one tuple nor those in different tuples are assumed to be distinct. Such a conjunction is said to be *satisfiable* iff there exists an assignment of elements of $\text{DOM}_{\mathcal{D}}$ to the variables such that the conjunction becomes true in $\mathcal{D}$.

For example, let $p_1(x,y)$ be the predicate $\exists z(x + z^2 = y)$ in $pred(\mathcal{R})$, and let $p_2(x,y)$ be the predicate $x > y$ in $pred(\mathcal{R})$. Obviously, neither the conjunction $p_1(x,y) \wedge p_2(x,y)$ nor $p_2(x,x)$ is satisfiable.

Definition 3.4.2 *A concrete domain $\mathcal{D}$ is called* admissible *iff* (i) *the set of its predicate names is closed under negation and contains a name for* $\text{DOM}_{\mathcal{D}}$, *and* (ii) *the satisfiability problem for finite conjunctions of the above mentioned form is decidable.* □

The concrete domain $\mathcal{R}$ is admissible. This is a consequence of Tarski's decidability result for real arithmetic [Tarski, 1951; Collins, 1975]. For the linear case

[5] For the sake of simplicity it is assumed here that the formula itself is the predicate name. In applications, the user will probably take his own intuitive names for these predicates.

(where the polynomials in the equalities and inequalities have to be linear) there exist more efficient methods (see e.g. [Weispfenning, 1988; Loos and Weispfenning, 1990]). Another important concrete domain is $\mathcal{R}_{lp}$ *rational numbers with inequalities between linear polynomials.* For $\mathcal{R}_{lp}$ efficient, incremental reasoning algorithms are available (see for example [Jaffar *et al.*, 1990; Jaakola, 1990]).

3.4.2 The Additional Operators

With the above formalization of concrete domains the extension ALCFP($\mathcal{D}$) of ALCF, which is parametrized by an admissible concrete domain $\mathcal{D}$, can be defined. The new concept forming operators can be seen as generalizations of the value restriction and the exists-in restriction.

Definition 3.4.3 (syntax of ALCFP($\mathcal{D}$)) *The concept formalism of* ALCF *is extended by the following operators. Let $u_1, \cdots, u_n$ be role/attribute chainings. Then*

$\forall u_1, \cdots, u_n.\rho$ (generalized value restriction)
$\exists u_1, \cdots, u_n.\rho$ (generalized exists-in restriction)

are concept terms in each of the following cases: The term ρ, which is called restrictor,

1. *is a predicate of the concrete domain with arity n,*
2. *is of the form p or $\neg p$, where p is an abstract predicate of arity n,*
3. *is a concept term and $n = 1$, or*
4. *is "$=$" or "$\neq$", n is 2, and u_1, u_2 are chainings of* attributes. □

Since the concrete domain extends the abstract domain $\text{DOM}_\mathcal{I}$, the interpretation of all constructs in ALCF has to be reconsidered. This somehow makes the definition of the semantics complicated at first glance.

Definition 3.4.4 (semantics of ALCFP($\mathcal{D}$)) *The differences of interpretations of* ALCF *and the extended language are as follows:*

The set $\text{DOM}_\mathcal{I}$, which is called abstract domain *for this language, is required to be disjoint to $\text{DOM}_\mathcal{D}$. Because attributes and roles link the abstract with the concrete domain their interpretation is liberated: An attribute f is interpreted as a partial function*

$$f^\mathcal{I} : \text{DOM}_\mathcal{I} \longrightarrow \text{DOM}_\mathcal{I} \cup \text{DOM}_\mathcal{D}$$

and a role r as a binary predicate

$$r^\mathcal{I} \subseteq \text{DOM}_\mathcal{I} \times (\text{DOM}_\mathcal{I} \cup \text{DOM}_\mathcal{D}).$$

An abstract predicate p of arity n is interpreted as $p^\mathcal{I} \subseteq \text{DOM}_\mathcal{I}^n$ and $(\neg p)^\mathcal{I}$ as $\text{DOM}_\mathcal{I}^n \backslash p^\mathcal{I}$, and a concrete predicate p is interpreted as $p^\mathcal{I} := p^\mathcal{D}$. It remains to define how the new operators are interpreted:

$a \in (\forall u_1, \cdots, u_n.\rho)^{\mathcal{I}}$ *iff*
for all $b_1, \cdots, b_n$ *with* $(a, b_1) \in u_1^{\mathcal{I}}$, $\cdots$, $(a, b_n) \in u_n^{\mathcal{I}}$ *we have* $(b_1, \cdots, b_n) \in \rho^{\mathcal{I}}$

$a \in (\exists u_1, \cdots, u_n.\rho)^{\mathcal{I}}$ *iff*
there exists $b_1, \cdots, b_n$ *with* $(a, b_1) \in u_1^{\mathcal{I}}$, $\cdots$, $(a, b_n) \in u_n^{\mathcal{I}}$ *and* $(b_1, \cdots, b_n) \in \rho^{\mathcal{I}}$

□

Analogous to ALCF the satisfiability problem and the subsumption problem in ALCFP($\mathcal{D}$) are interreducible. The satisfiability of a concept C can be reduced to the consistency of an assertion $a : C$ (cf. next section) and using Theorem 3.5.7 one gets:

Theorem 3.4.5 (T-box reasoning in ALCFP($\mathcal{D}$)) *There exist decision procedures for the satisfiability and the subsumption problem in* ALCFP($\mathcal{D}$). □

Since the subsumption algorithm is the crucial subroutine of a concept classification algorithm, the theorem implies that a concept classifier can be (and has been) implemented for this concept language.

3.5 The Assertional Box

The *assertional box (A-box)* is a formalism to make assertions about objects using the roles, attributes, concepts and predicates introduced in the terminology. Throughout this section it is assumed that a terminology $\mathcal{T}$ is given. For example, let father be a concept in $\mathcal{T}$ and let child be a role. Then the assertions John : father, (John, Boy) : child, (John, Bob) : child say that John is a father who has a child Boy and a child Bob.

Many systems adopt a *unique name assumption (UNA)* for the names of the objects. In the example this would imply that Boy and Bob denote two different objects. Since attributes and, in particular, attribute (dis)agreements belong to the concept language of ALCFP($\mathcal{D}$), the assertional formalism considered here, will NOT adopt the UNA. Instead, explicit equality and negated equality is introduced. In the example, without UNA, later reasoning based on more assertions could reveal that Boy and Bob are in fact the same person (perhaps his name is Bob but his mother always calls him Boy).

When the rule scheme of Chapter 5 is applied to the terminological formalism discussed here, there will be a global collection of assertions (a fact base) and various other little 'A-boxes' as premises or consequences of rules. Generic individuals in premises or consequences of rules will typically be universally quantified on rule level. Objects occurring in the fact base are existentially quantified on A-box level. In Section 5.2 this view of quantification will be refined.

Definition 3.5.1 (A-box syntax) *Let be* Ob *an alphabet of* object names, *a, b, possibly with indices names from* Ob, *C a concept, R a role or attribute, P_c a concrete*

n-ary predicate, and P_a an abstract n-ary predicate. Then the following expressions are A-box assertions*:*

$a : C$	(membership assertion)
$(a, b) : R$	(role-/attribute-filler assertion)
$(a_1, \cdots, a_n) : P_c$	(concrete predicate assertion)
$(a_1, \cdots, a_n) : P_a$	(abstract predicate assertion)
$(a_1, \cdots, a_n) : \neg P_a$	(negated abstract predicate assertion)
$a = b$	(equality assertion)
$a \neq b$	(negated-equality assertion)

To avoid unnecessary case distinctions each of these assertions is often written as a generalized membership assertion $x : \rho$ *where x is a (possibly degenerated) tuple of objects and ρ is a restrictor (i.e., one of C, R, P_c, P_a, $\neg P_a$, $=$, $\neq$ with the appropriate arity).*
Assertional formulas *are inductively defined:*

- *Every A-box assertion is an (assertional) formula.*
- *Let A and B be (assertional) formulas and let x be a variable. Then $A \wedge B$, $\sim A$,[6] and $\exists x(A)$ are formulas.*

An A-box *is an assertional formula of the form*

$$\exists y_1 \cdots \exists y_n (x_1 : \rho_1 \wedge \cdots \wedge x_m : \rho_m)$$

where y_i, $i = 1, \cdots, n$, are the variables occurring in the generalized assertions $x_j : \rho_j$, $j = 1, \cdots, m$. □

An A-box is identified with the set of its A-box assertions. Note that each A-box is an assertional formula but not vice versa. The semantics of an assertional formula is defined by extending the notion of an interpretation of a terminology.

Definition 3.5.2 (A-box semantics) *Let a terminology $\mathcal{T}$ and an assertional formula A be given. Then an* interpretation $\mathcal{I}$ of A (with respect to $\mathcal{T}$) *is a model of $\mathcal{T}$. An* (object) assignment α *is a partial function from* Ob *to* $\text{DOM}_\mathcal{I} \cup \text{DOM}_\mathcal{D}$. *As usual, α is identified with the induced natural homomorphism on the term and formula structure.*

An interpretation $\mathcal{I}$ and an assignment α satisfy *an assertional formula according to the following inductive definition. If a formula A is satisfied by $\mathcal{I}, \alpha$, this is abbreviated as $\mathcal{I}, \alpha \models A$.*

- $\mathcal{I}, \alpha \models x : \rho$ *if* $x\alpha \in \rho^\mathcal{I}$.
- $\mathcal{I}, \alpha \models A \wedge B$ *if* $\mathcal{I}, \alpha \models A$ *and* $\mathcal{I}, \alpha \models B$.

[6] For negation the symbol $\sim$ is used instead of the usual $\neg$, because the latter denotes already the complement operator with respect to the abstract domain $\text{DOM}_\mathcal{I}$ in the concept language.

- $\mathcal{I}, \alpha \models \sim A$ *if not* $\mathcal{I}, \alpha \models A$.
- $\mathcal{I}, \alpha \models \exists x(A)$ *if there is a* $d \in \mathrm{DOM}_{\mathcal{I}} \cup \mathrm{DOM}_{\mathcal{D}}$ *and a fresh variable* y *such that* $\mathcal{I}, \alpha[y \mapsto d] \models A[x \mapsto y]$.[7]

An interpretation $\mathcal{I}$ *is a* model of A (with respect to $\mathcal{T}$) *if for all assignments* α *the pair* $\mathcal{I}$, α *satisfies* A. *An assertional formula is* consistent *if it has a model. An assertional formula is* valid *if each interpretation is a model.* □

The logical connectives $\vee$ and $\Rightarrow$ as well as the quantifier $\forall$ are used to abbreviate (assertional) formulas in the usual way.

The services that are going to be defined are restricted to A-boxes. Without these restrictions the corresponding reasoning problems would be undecidable.

Definition 3.5.3 (A-box services) *For a terminology* $\mathcal{T}$ *and an A-box* $\mathcal{A}$ *the following services are defined:*

Consistency test: *Checks whether the A-box is consistent.*

(Generalized) membership test: *Let* $x : \rho$ *be a generalized membership assertion such that the objects in* x *occur in* $\mathcal{A}$. *Then the tuple* x *is a* generalized member *of* ρ (with respect to $\mathcal{A}$ and $\mathcal{T}$) *if* $\forall x(\mathcal{A}' \Rightarrow x : \rho)$ *is valid where* $\mathcal{A}'$ *is the assertional formula that is obtained from* $\mathcal{A}$ *by omitting the existential quantifiers belonging to the variables in the tuple* x.

A-box subsumption: *Let* $\mathcal{B}$ *be a second A-box and let* z *be an* n*-ary tuple of pairwise different variables occurring in* $\mathcal{B}$. [8] *Then* $\mathcal{B}$ *subsumes* $\mathcal{A}$ *with respect to* x *if* $\forall x(\mathcal{A}' \Rightarrow \mathcal{B}')$ *is valid. Here* $\mathcal{A}'$ *(resp.* $\mathcal{B}'$*) is the assertional formula that is obtained from* $\mathcal{A}$ *(resp.* $\mathcal{B}$*) by omitting the quantifiers belonging to the variables in* x.

Object classification: *Let* a *be an object occurring in* $\mathcal{A}$. *Then the* object classification *(service) computes a* realization of a *which is a set* M *of concept names satisfying the following requirements:*

- *The object* a *is a member of* C, *for all* $C \in M$.
- *If there is a concept name* C *such that* a *is a member of* C, *then there exists* $C' \in M$ *such that* C *subsumes* C'.
- *If* $C \in M$ *and* $C' \in M$ *such that* $C \neq C'$, *then* C *does not subsume* C' *nor does* C' *subsume* C

---

[7] The assignment $\alpha[y \mapsto d]$ is defined by $z \longmapsto \begin{cases} d, & \text{if } z = y \\ z\alpha, & \text{otherwise,} \end{cases}$ and $A[x \mapsto y]$ denotes the assertional formula A with all occurrences of x replaced by y.

[8] These are not necessarily all variables occurring in $\mathcal{B}$.

Informally speaking, M is the set of most specific concept names a belongs to (where for equivalent concepts only one representative is selected). □

Setting $\mathcal{B} := \exists x(x : \rho)$ we see that A-box subsumption is a generalization of the generalized membership test. It is known that for ALCF with attribute agreements and disagreements it is undecidable whether for a concept C there is an interpretation such that C is interpreted as the whole domain (Theorem 5.3 in [Baader *et al.*, 1991]). Setting $\mathcal{A} := \emptyset$, $n = 0$, and $\mathcal{B} := \exists x(x : \sim C)$ this problem can be reduced to A-box subsumption [9] and, thus, one gets:

Proposition 3.5.4 *A-box subsumption for* ALCF *extended by attribute agreements and disagreements is undecidable. Since* ALCFP($\mathcal{D}$) *is an extension of* ALCF, *A-box subsumption is undecidable for* ALCFP($\mathcal{D}$), *too.* □

It is also possible to give a reduction with $n > 0$. The real problem is that $\exists x(x : \sim C)$ is not related to the assertions in $\mathcal{A}$ so that the implication can only hold if $\exists x(x : \sim C)$ is valid. This provides a first idea for an appropriate restriction of the general problem.

Definition 3.5.5 (rooted A-boxes) *Let $\mathcal{A}$ be an A-box. Then an object a is* directly linked (by attributes) *to an object b if there is an assertion $(a, b) : R$ in $\mathcal{A}$, R a role or attribute (resp. R an attribute).* Linked (by attributes) *is the reflexive, transitive closure of directly linked (by attributes). The A-box $\mathcal{A}$ is* rooted by objects $x_1, \cdots, x_n$ *if*

1. *for each object b in $\mathcal{A}$ there exists an $a \in \{x_1, \cdots, x_n\}$ such that a is linked to b and*

2. *for each assertion $a \neq b$ in $\mathcal{A}$ there are $x, y \in \{x_1, \cdots, x_n\}$ which are not necessarily distinct such that x is linked by attributes to a and y is linked by attributes to b.*[10]

□

Let $\mathcal{A}$ and $\mathcal{B}$ be two A-boxes such that $y_1, \cdots, y_n$ are objects in $\mathcal{A}$, and $\mathcal{B}$ is rooted by $y_1, \cdots, y_n$. Then the problem whether $\mathcal{B}$ subsumes $\mathcal{A}$ with respect to $y_1, \cdots, y_n$ is called the *A-box subsumption problem for rooted A-boxes.*

T-box and A-box services can be reduced to a consistency test for (a restricted form of) assertional formulas. In particular, a concept C is satisfiable iff the A-box $c : C$ is consistent. The A-box $\mathcal{B}$ subsumes $\mathcal{A}$ with respect to $y_1, \cdots, y_n$ if $\exists y_1, \cdots, y_n(\mathcal{A}' \wedge \sim \mathcal{B}')$ is not consistent.

An A-box where some of the objects are not existentially quantified and occur free is called an *A-box formula.* A *generalized A-box (GA-box)* is an A-box $\mathcal{B}$ (considered as a set) with free objects $y_1, \cdots, y_n$ that root $\mathcal{A}'$ and occur in $\mathcal{B} \setminus \{\sim \mathcal{A}'\}$.

[9] Note, $\emptyset \Rightarrow \exists x(x : \sim C)$ iff $\forall x(x : \sim\sim C)$ is inconsistent.

[10] It is not clear whether Theorem 3.5.7 would still hold if this condition would be dropped.

Observation 3.5.6 *Satisfiability and subsumption of concepts as well as consistency, membership, and A-box subsumption for A-boxes can be reduced to a consistency tests for generalized A-Boxes.* □

In the next section a consistency test for generalized A-boxes is presented. Proposition 3.7.1 implies that this algorithm is a decision procedure for this consistency problem. Together, with the observation this implies the following theorem.

Theorem 3.5.7 (A-box reasoning in ALCFP($\mathcal{D}$)) *There exist decision procedures for the consistency problem, the (generalized) membership problem, and the A-box subsumption problem for rooted A-boxes.* □

Because the membership test is the crucial operation for implementing the object classification service, the theorem implies that there is also an algorithm that computes in finite time for a given A-box, a terminology, and an object the realization of the object.

For the next section the following technical remark is needed. Let $\mathcal{B}$ and $\sim\mathcal{A}'$ be as above. Then $\sim\mathcal{A}'$ is equivalent to an assertional formula

$$\forall z[(\bigwedge_i (a_i, b_i) : R_i) \Rightarrow \bigvee_j A_j]$$

satisfying the following requirements:

1. The R_i are roles or attributes.
2. All objects in $\bigvee_j A_j$ occur in $\mathcal{B} \setminus \{\sim\mathcal{A}'\}$ or $\bigwedge_i (a_i, b_i) : R_i$.
3. Each A_j is of the form $\sim(x : \rho)$ where $x : \rho$ is a generalized membership assertion.

Such a formula is referred to as the implication normal form of $\sim\mathcal{A}'$ (with respect to $\mathcal{B}$).

3.6 The Basic Algorithm

This section presents an algorithm that decides in finite time whether a given GA-box $\mathcal{A}$ is consistent. The algorithm is a generalization of the technique that was introduced in [Schmidt-Schauß and Smolka, 1991] and further elaborated, e.g., in [Baader and Hanschke, 1991b; Baader, 1991; Hollunder *et al.*, 1990]

Roughly, the algorithm proceeds as follows. It starts with a given GA-box $\mathcal{A}$, and applies transformation rules to $\mathcal{A}$ that make the knowledge represented by the assertions more explicit. Ultimately, one of the following two situations occurs:

1. The GA-box becomes "obviously contradictory", or

2. all knowledge has been made explicit.

In the latter case the GA-box is called *complete* and induces a model of the original $\mathcal{A}$. In the other case $\mathcal{A}$ is inconsistent.

Sometimes it is necessary to make a case distinction during the transformation process, since disjunctions occur (implicitly and explicitly) in the formalism. Hence, a transformation step may replace a single GA-box $\mathcal{A}$ by new GA-boxes $\mathcal{B}_1, \cdots, \mathcal{B}_n$, $n > 1$. In this case $\mathcal{A}$ is inconsistent if all $\mathcal{B}_i$, $1 \leq i \leq n$ are inconsistent. For that reason, the algorithm operates with sets of GA-boxes rather than a single GA-box. If the consistency of an GA-box $\mathcal{A}$ has to be checked, the algorithm is initialized with the singleton set $\mathcal{M}_0 = \{\mathcal{A}_0\}$ where $\mathcal{A}_0$ is a normalized version of $\mathcal{A}$. The following subsection (3.6.1) describes this normalization.

3.6.1 Unfolding and Implication Normal Form

Let a terminology $\mathcal{T}$ and an A-box $\mathcal{A}_0$ be given. To simplify the presentation of the algorithm, the GA-box is first normalized by the *unfolding rule*. It replaces a concept name C by its definition t if the concept definition $C = t$ is in $\mathcal{T}$. Because terminologies do not contain cycles this rule can only be applied finitely many times. After defined concepts have been replaced, the terminology is not needed any more for the consistency test. If a negated A-box occurs in $\mathcal{A}_0$, it is replaced by its implication normal form.

3.6.2 Transformation Rules

This section presents the transformation rules that operate on the set $\mathcal{M}_0$. They generate a finite sequence (see the next section for a proof of the finiteness) of sets $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, ... $\mathcal{M}_k$ of finite GA-boxes. The letters s and t denote concept terms, ρ is a restrictor, and the R_i are attributes or roles.

The rules operate on the level of assertions. For these rules an expression of the form

$$\frac{\textit{premises}}{\textit{consequences}}$$

has to be read as follows: if there is an A-box $\mathcal{A}$ in the current $\mathcal{M}_i$ that fulfills the premises, then the successor $\mathcal{M}_{i+1}$ is obtained by adding the appropriately instantiated consequences to $\mathcal{A}$.

If vertical bars "|" occur in the consequence of a rule, this means that the A-box $\mathcal{A} \in \mathcal{M}_i$ to which the rule is applied has to be replaced with new A-boxes for each of the alternatives that are separated by the bar(s). Hence, in these cases $\mathcal{M}_{i+1}$ contains more A-boxes than its predecessor $\mathcal{M}_i$.

3.6.2.1 Pushing Negation

The *negation rules* propagate negation ("$\neg$") towards the leaves of the concept terms. Recall that $\neg$ is a complement operator with respect to $\mathrm{DOM}_{\mathcal{I}}$ and that attributes and roles link the abstract domain with the concrete domain. It is convenient to use the negation operator $\sim$ also as a *global complement operator* for restrictors by extending its definition by $\sim\rho^{\mathcal{I}} = (\mathrm{DOM}_{\mathcal{I}} \cup \mathrm{DOM}_{\mathcal{D}})^n \setminus \rho^{\mathcal{I}}$ where ρ is a restrictor with arity n (Definition 3.4.3).

$$\frac{a : \neg\neg s}{a : s} \qquad \frac{a : \neg(s \sqcap t)}{a : \neg s \sqcup \neg t} \qquad \frac{a : \neg(s \sqcup t)}{a : \neg s \sqcap \neg t}$$

$$\frac{a : \neg\forall v_1, \cdots, v_n.\rho}{a : \exists v_1 \cdots v_n.\sim\rho} \qquad \frac{a : \neg\exists v_1, \cdots, v_n.\rho}{a : \forall v_1 \cdots v_n.\sim\rho}$$

where the v_i are attribute/role chainings.

3.6.2.2 The $\sim$ Rules

The following rules deal with the global complement operator if it occurs at the top level in an assertion.

$$(\mathbf{R}{\sim\sim}) \quad \frac{a : \sim\sim\rho}{a : \rho}$$

$$(\mathbf{R}{\sim}\mathcal{D}) \quad \frac{(a_1, \cdots, a_n) : \sim\rho}{a_1 : \top \mid \cdots \mid a_n : \top \mid (a_1, \cdots, a_n) : \overline{\rho}}$$

if ρ is a concrete predicate and $\overline{\rho}$ is the complement of ρ with respect to $\mathrm{DOM}_{\mathcal{D}}$ (since $\mathcal{D}$ is admissible $\overline{\rho}$ is also a predicate of the concrete domain), and $\top$ is a specific concept name that is always interpreted as $\mathrm{DOM}_{\mathcal{I}}$.

$$(\mathbf{R}{\sim}\mathcal{P}) \quad \frac{(a_1, \cdots, a_n) : \sim\rho}{a_1 : \mathcal{D} \mid \cdots \mid a_n : \mathcal{D} \mid (a_1, \cdots, a_n) : \neg\rho}$$

if ρ is a concept term and $n = 1$ or if it is an abstract predicate.

$$(\mathbf{R}{\sim}{=}) \quad \frac{(a_1, a_2) : \sim{=}}{(a_1, a_2) : \neq}$$

$$(\mathbf{R}{\sim}{\neq}) \quad \frac{(a_1, a_2) : \sim{\neq}}{(a_1, a_2) : =}$$

The case $a : \neg\sim\rho$ does not occur, because the algorithm is applied to GA-boxes only.

3.6.2.3 The Operator Rules

These rules split concept terms into their immediate subterms and generate new assertions.

$$(\mathbf{R}\sqcap)\ \frac{a : s \sqcap t}{a : s, a : t}$$

$$(\mathbf{R}\sqcup)\ \frac{a : s \sqcup t}{a : s \mid a : t}$$

This rule replaces two A-boxes for each A-box the rule is applied to.

$$(\mathbf{R}\exists)\ \frac{a : \exists v_1 \cdots v_n.\rho}{(a, b_1) : v_1, \cdots, (a, b_n) : v_n, (b_1, \cdots, b_n) : \rho}$$

Here the b_i are fresh individual constants.

$$(\mathbf{R}\forall)\ \frac{(a, b_1) : v_1, \cdots, (a, b_n) : v_n, a : \forall v_1 \cdots v_n.\rho}{(b_1, \cdots, b_n) : \rho}$$

A premise $(a, b) : v$ is fulfilled by objects x, y if

1. $x = y$ and v is ϵ or
2. there is $(x, z) : R$ in the A-box, v is of the form $R \circ v'$ where R is an attribute or role, and, recursively, z, y fulfill $(c, b) : v'$ is fulfilled, for some c.

$$(\mathbf{R}\Rightarrow)\ \frac{(a_1, b_1) : R_1, \cdots, (a_n, b_n) : R_n, \mathcal{A}}{x_1\tau : {\sim}\rho_1 \mid \cdots \mid x_k\tau : {\sim}\rho_k}$$

where $\mathcal{A}$ is a negated A-box formula

$$\forall z[(\bigwedge_{i=1\cdots n} (y_{1i}, y_{2i}) : R_i) \Rightarrow ({\sim}x_1 : \rho_1 \vee \cdots \vee {\sim}x_k : \rho_k)]$$

in implication normal form and τ is a substitution such that $(y_{1i}, y_{2i})\tau = (a_i, b_i)$, for all i, $1 \leq i \leq n$. This rule replaces each affected GA-box by k GA-boxes.

3.6.2.4 The Role and Attribute Rules

The (R$\exists$) rule may generate new assertions of the form $(a, b) : v$ where v is a chaining of attributes or roles. It may also cause *forks*. These are pairs of attribute-filler assertions $(a, b) : f$, $(a, c) : f$ with $b \neq c$. These expressions are treated by the following rules:

$$(\mathbf{R}\circ)\ \frac{(a, b) : R \circ v}{(a, c) : R, (c, b) : v}$$

Here c is a fresh individual.

$$(\mathbf{R}\epsilon)\ \frac{(a,b):\epsilon}{a=b}$$

$$(\mathbf{R}\mapsto)\ \frac{(a,b_1):f,(a,b_2):f}{b_1=b_2}\ \text{if } f \text{ is an attribute.}$$

3.6.2.5 The Identification Rule

The attribute agreements and the functional character of the attributes may lead to equality assertions. These are treated by the following rule:

$$(\mathbf{R}=)\ \frac{(a,b):=}{\text{replace } a \text{ by } b \text{ in the affected GA-box}}$$

3.6.2.6 The Domain Rules

The abstract and the concrete domain are disjoint. This may lead to obvious contradictions. The domain rules try to make explicit the domain to which an individual belongs.

$$(\mathbf{R}\mathcal{P}\top)\ \frac{(a_1,\cdots,a_n):\rho}{a_1:\top,\cdots,a_n:\top}$$

if ρ is a primitive concept or an abstract predicate or a negated abstract predicate.

$$(\mathbf{R}R\top)\ \frac{(a_1,a_2):R}{a_1:\top}\ \text{if } R \text{ is a role or attribute.}$$

$$(\mathbf{R}\mathcal{D}\top)\ \frac{(a_1,\cdots,a_n):\rho}{a_1:\mathcal{D},\cdots,a_n:\mathcal{D}}$$

if ρ is a concrete predicate different from $\mathcal{D}$.

3.6.3 Obvious Contradictions

A single GA-box $\mathcal{A}$ is *obviously contradictory* in each of the following cases:

Primitive Clash: The GA-box contains a pair of assertions of the form $x:\rho, x:\neg\rho$ where ρ is an abstract predicate (resp. a concept term) and x is a tuple of objects (resp. a single object).

Domain Clash: The GA-box contains $a:\top$, $a:\mathcal{D}$.

Equality Clash: The GA-box contains $a \neq a$.

Concrete Domain Clash: The GA-box contains predicate assertions $x_1:p_1, \ldots, x_n:p_n$ where the p_i are concrete predicates and the satisfiability test of the concrete domain says that the conjunction of these predicates is not satisfiable.

3.6.4 The Strategy

In order to get a terminating algorithm the rule application has to be restricted. Identifications of objects have to take place as soon as possible. So the role and attribute rules (3.6.2.4) and the identification rules (3.6.2.5) are executed with the highest priority. A GA-box to which none of these rules is applicable is called *RI-reduced.*

A rule is applied at most once per set of instances. Some objects in assertions are replaced during applications of the (R=) rule. Transformation rules must not be applied again to these assertions (although the premises are not exactly the same). If the GA-box contains an obvious contradiction (see Section 3.6.3), rules must not be applied, either.

If the v_i in an assertion $a : \forall v_1 \cdots v_n.\rho$ are all attribute chainings, the (R$\forall$) rule is applied at most once to this assertion. Applications of the (R$\forall$) rule to assertions that are not of this form and applications of the (R$\Rightarrow$) rule are called *foreign.* Foreign applications take place only if the assertion that would be added is not present.

3.6.5 Summary of Algorithm

Figure 3.5 summarizes the consistency test of ALCFP($\mathcal{D}$) using a pseudo programming language. The procedure takes a GA-box $\mathcal{A}$ as an argument and checks whether it is consistent or not.

```
define procedure check-consistency(A)
  A_0 := unfolded-implication-normal-form(A)
  r := 0
  M_0 := {A_0}
  while 'a transformation rule is applicable to M_r' do
    r := r + 1
    M_r := apply-a-transformation-rule(M_{r-1})
  endwhile
  if 'there is an A ∈ M_r that is not obviously contradictory' then
    consistent
  else
    inconsistent
  endif
```

Figure 3.5: Consistency Test of ALCFP($\mathcal{D}$)

3.7 The Proof

In this section termination, soundness, and completeness of the consistency test (Figure 3.5) are proved. Together, these facts imply that the algorithm is a decision procedure for the consistency of an GA-box $\mathcal{A}$.

Proposition 3.7.1 *Assume that the procedure 'check-consistency' (Figure 3.5) is applied to $\mathcal{A}$. Then*

1. *the algorithm always computes in finite time a set $\mathcal{M}_r$ of GA-boxes each of which is complete or obviously contradictory, and*
2. *the initial GA-box is inconsistent iff all GA-boxes $\mathcal{A} \in \mathcal{M}_r$ are obviously contradictory.*

Proof. The proposition is a consequence of three lemmata (3.7.2, 3.7.4, 3.7.5) stated and proved below. □

Unfolding and transformation into implication normal form terminate and do not change the consistency of an GA-box. Hence, these preparatory steps are neglected in the remainder of the proof. *RI*-reduction, which also terminates and does not change the satisfiability of an GA-box, is considered to be built into the underlying data structure. I.e., if any rule is applied, RI-reduction is performed immediately.

The while loop of the algorithm reduces the semantic problem of consistency for the GA-box $\mathcal{A}_0$ to a simple almost syntactic[11] problem for a finite set $\mathcal{M}_r$ of GA-boxes. This syntactic problem is to check whether there is an GA-box in $\mathcal{M}_r$ that is not obviously contradictory. In order to show the correctness of the reduction, termination is proved first.

Assume that a computation using the algorithm is given and that in a single execution of the loop body the RI-reduced GA-boxes $\mathcal{B}_1,\ldots,\mathcal{B}_n$, $n > 0$, have been derived by an application of one of the transformation rules to an RI-reduced GA-box $\mathcal{B}$. Then the $\mathcal{B}_i$ are called *descendants* of $\mathcal{B}$.

Lemma 3.7.2 (termination) *The algorithm always computes a complete set of GA-boxes $\mathcal{M}_r$ in finite time.*

Proof. Assume that a possibly infinite computation is given. In order to show termination it suffices to prove that there is an infinite chain of GA-boxes $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \cdots$ where $\mathcal{A}_{i+1}$ is a descendant of $\mathcal{A}_i$.

Assume to the contrary that there is such an infinite sequence. Each $\mathcal{A}_i$ will be mapped to an element $\Psi(\mathcal{A}_i)$ of a set Q which is equipped with a well-founded strict partial ordering $\gg$. Since the ordering is well-founded, i.e., has no infinitely decreasing chains, a contradiction is obtained as soon as the following lemma has been established.

[11] The problem is syntactic as far as the concrete domain is not concerned.

Lemma 3.7.3 *If $\mathcal{A}'$ is a descendant of $\mathcal{A}$, then $\Psi(\mathcal{A}) \gg \Psi(\mathcal{A}')$.*

The elements of the set Q will have a rather complex structure. They are finite multisets of 3-tuples; the first and second component of the 3-tuple being natural numbers, and the third being a multiset of natural numbers.

Multisets are like sets, but allow multiple occurrences of identical elements. For example, $\{2,2,2\}$ is a multiset which is distinct from the multiset $\{2\}$. A given ordering on a set T can be extended to form an ordering on the finite multisets over T. In this ordering, a finite multiset M is larger than a finite multiset M' iff M' can be obtained from M by replacing one or more elements in M by any finite number of elements taken from T, each of which is smaller than one of the replaced elements. For example, $\{2,2,2\}$ is larger than $\{2\}$ and $\{2,2,1,1,0\}$. [Dershowitz and Manna, 1979] show that the induced ordering on finite multisets over T is well-founded if the original ordering on T is so.

The nonnegative integer components of the 3-tuples are compared with respect to the usual ordering on integers. Whole tuples are ordered lexicographically from left to right, for example, $(c_1, ..., c_3)$ is larger than $(c'_1, ..., c'_3)$ iff there exists $i, 1 \leq i \leq 3$, such that $c_1 = c'_1$, ..., $c_{i-1} = c'_{i-1}$, and c_i is larger than c'_i. If the orderings on the components are well-founded, the lexicographical ordering on the tuples is also well-founded.

The finite multisets of the 3-tuples are compared with respect to the multiset ordering induced by this lexicographical ordering. This multiset ordering is the well-founded ordering $\gg$ on Q mentioned above.

Before we can define the mapping Ψ from GA-boxes to elements of Q, we need a few more definitions and observations.

The *size* of a term or assertion is inductively defined as follows:

1. $|x : \rho| := |\rho|$

2. The size $|v|$ of a role/attribute chain $v = R_1 \circ \ldots \circ R_n$, $n \geq 0$, is its length n.
   $|\langle \textit{concrete predicate} \rangle| := 1$, $|\langle \textit{abstract predicate} \rangle| := 1$,
   $|=| := 1$, $|\neq| := 1$,
   $|\langle \textit{concept name} \rangle| := 1$

3. $|t_1 \sqcap t_2| := |t_1| + |t_2|$, $|t_1 \sqcup t_2| := |t_1| + |t_2|$, $|{\sim}\rho| := 3 * |\rho|$, $|\neg\rho| := 2 * |\rho|$
   $|\exists v_1 \cdots v_n.\rho| := |\rho| + \max\{|v_1|, \cdots, |v_n|\}$,
   $|\forall v_1 \cdots v_n.\rho| := |\rho| + \max\{|v_1|, \cdots, |v_n|\}$

4. The size of a negated A-box in implication normal form

$$|\forall \underline{x}[(\bigwedge_{i=1..n} (a_i, b_i) : R_i) \Rightarrow ({\sim}y_1 : \rho_1 \vee \cdots \vee {\sim}y_m : \rho_m)]|$$

   is $n + \max\{|{\sim}\rho_1|, \cdots, |{\sim}\rho_m|\}$

The proof proceeds by establishing some upper bounds needed in the definition of the mapping Ψ.

1. If a transformation rule for pushing negation, a $\sim$ rule, or an operator rule is applied to a set of assertions S, then for each assertion $x : \rho$ that is added by this rule application there is an $x' : \rho' \in S$ such that $|\rho'| > |\rho|$.

2. There is an integer s_0 such that for all terms t occurring in a computation $s_0 > |t|$.

3. An *(attribute) cluster* of a GA-box $\mathcal{A}$ is a maximal set of objects Cl such that each pair of objects $a, b \in Cl$ is linked by an undirected path of attribute filler assertions in $\mathcal{A}$. For an object a in $\mathcal{A}$ $Cl\langle a\rangle$ denotes the unique cluster of $\mathcal{A}$ satisfying $a \in Cl\langle a\rangle$.

4. The initial GA-box contains already objects and clusters. These are called *old objects* and *old clusters*, respectively. Objects and clusters introduced during the computation are called *new*. If an old and a new object are identified the 'remaining' object is still termed old. If a cluster contains at least one old object, it is old.

   A close look at the transformation rules reveals that each new cluster Cl has exactly one *incoming edge* $(a, b) : R$ where R is a role, $b \in Cl$, and $a \notin Cl$.

5. The *distance* $d(a, b)$ *of two objects* a *and* b *(in a given A-box)* is the length of the shortest undirected path of filler assertions linking a to b.

6. The objects that occur free in the possibly existing negated A-box ϕ are all old. Let n_0 be the number of filler assertions occurring in the premise of ϕ. Then, if a new object b is affected by the application of the (R$\Rightarrow$) rule, $d(a, b) \leq n_0$, for some object a in ϕ.[12]

7. Let b be an object with $d(a, b) > n_0$, for every old object a. A close look at the transformation rules reveals that if there is a membership assertion $b : t$ (resp. if such an assertion does not exist), then there exists an assertion $a : t'$ such that

   (a) a is linked to b by a path of length $\leq s_0$ and

   (b) $|t'| > |t|$ (resp. $|t'| > 0$).

   (c) $a : t'$ has been introduced in the GA-box before $b : t$ (resp. b).

8. As a consequence of (6) and (7) there exists an integer $\nu(s_0, n_0)$ depending on s_0 and n_0 such that for each new object b there exists an old object a such that a is linked to b by a path of a length $< \nu(s_0, n_0)$.

[12] An object is affected by a rule application, if it occurs in the instantiated rule.

9. Each new cluster Cl is linked by a chaining of role-filler assertions and clusters to an old cluster Cl'

$$\begin{array}{rll} Cl' = & Cl_0, & (a'_0, a_1) : R_1, \\ & Cl_1, & (a'_1, a_2) : R_2, \\ & Cl_2, & \\ & \cdots & \\ & & (a'_{n-1}, a_n) : R_n, Cl_n = Cl \end{array}$$

where Cl_i is a new cluster, $a_i, a'_i \in Cl_i$, for $i = 1, \cdots, n-1$, $a'_0 \in Cl_0$, and $a_n \in Cl_n$. Since identifications take place only within a cluster or between objects from two old clusters (cf. Definition 3.5.5 (2)) the path is unique. The *generation* of the cluster Cl is the length n of the path.

10. Following (8) the 'generations' that occur in a computation are bounded by an integer g_0.

11. Let b be a new object. Then there is an $a \in Cl\langle b \rangle$ such that

    (a) there a is linked to b by attributes and

    (b) a is old or, otherwise, the unique incoming edge is of the form $(c, a) : R$.

12. The fact that in an RI-reduced GA-box there is at most one attribute filler per object and attribute together with (8,10) implies that the number of objects in a single cluster is bounded.

13. According to the strategy (3.6.4) and (12) for a cluster Cl only finitely many, foreign member-ship assertions $b : t$ can be asserted by the (R$\forall$) rule or the (R$\Rightarrow$) rule. Recall that an assertion is called *foreign* if it comes from the (R$\Rightarrow$) rule or from an application of the (R$\forall$) rule where a role occurs in a role/attribute chaining of the value restriction. Let f_0 be an upper bound of the number of foreign assertions that can occur for a single cluster.

On the basis of the upper bounds just derived the mapping $\Psi : \mathcal{A} \longmapsto \Psi A$ can be defined. Each cluster Cl is mapped to a 3-tuple $\psi(Cl)$ with the following components:

1. $g_0 - \langle$generation of $Cl\rangle$
2. $f_0 - \sharp$foreign assertions made to Cl
3. The multiset of integers $|x : \rho|$ where all objects in x belong to Cl, ρ is a value restriction not comprising a role, or it is not a value restriction and has not yet been processed by a rule pushing negation, an operator rule, nor a $\sim$ rule.[13]

[13]Note that if x comprises objects from different clusters, then $x : \rho$ is a predicate assertion or a negated abstract predicate assertion. These assertions do not cause any problems with respect to to termination.

Finally, $\Psi\mathcal{A}$ is defined as the multiset of 3-tuples $\psi(Cl)$ where Cl is a cluster in $\mathcal{A}$.

It follows a case analysis of the rule applications verifying Lemma 3.7.3. Let $\mathcal{A}'$ be a RI-reduced immediate descendant of $\mathcal{A}$ and let $\psi_{\mathcal{A}}(Cl)$ (resp. $\psi_{\mathcal{A}'}(Cl)$) be the image of a cluster with respect to $\mathcal{A}$ (resp. $\mathcal{A}'$).

Case analysis of rule application:

a) Pushing Negation (3.6.2.1): Each application of one of these rules affects only one cluster Cl. The first and the second component of $\psi_{\mathcal{A}}(Cl)$ and $\psi_{\mathcal{A}'}(Cl)$ are the same. The third component gets smaller, because one integer is replaced by one smaller integer. As the tuples are compared lexicographically the new tuple is smaller and $\Psi\mathcal{A} \gg \Psi\mathcal{A}'$.

b) The Operator Rules (3.6.2.3): An application of the (R$\sqcap$) or (R$\sqcup$) rule affects only a single cluster, too, and similar as in (a) we deduce $\Psi\mathcal{A} \gg \Psi\mathcal{A}'$.

   If an application of the (R$\forall$) rule to a member-ship assertion $a : \forall v_1 \cdots v_n.\rho$ does not generate foreign assertions, $|a : \forall v_1 \cdots v_n.\rho|$ is removed and the smaller $|\rho|$ is added to the multiset in the third component. The first and second component as well as the other 3-tuples are not changed and thus $\Psi\mathcal{A} \gg \Psi\mathcal{A}'$. If the application of the (R$\forall$) rule is foreign or if the (R$\Rightarrow$) rule is applied, the second component of another cluster decreases and the other 3-tuples are not changed. Hence $\Psi\mathcal{A} \gg \Psi\mathcal{A}'$.

   Consider the application of the (R$\exists$) rule to an assertion $a : \exists v_1 \cdots v_n.\rho$. If there is an i such that v_i contains one or more roles, new clusters $\{Cl_j\}_{j\in J}$, J finite and non-empty, are introduced. But these clusters have a greater generation and, thus, the first component of the new 3-tuples of these clusters is smaller than the 3-tuple of $Cl\langle a\rangle$. The third component of the 3-tuple of $Cl\langle a\rangle$ decreases and, thus, $\Psi\mathcal{A}'$ is obtained from $\Psi\mathcal{A}$ by replacing $\psi_{\mathcal{A}}(Cl\langle a\rangle)$ by the smaller $\psi_{\mathcal{A}'}(Cl\langle a\rangle)$ and $\psi_{\mathcal{A}'}(Cl_j)$, $j \in J$.

c) The $\sim$ rules are treated similarly to the (R$\sqcup$) rule.

Finally, the domain rules generate only new assertions to which none of the transformation or rewrite rules is applicable. This completes the proof of Lemma 3.7.2. □

To prove the second part of Proposition 3.7.1, the notion of *contradictory GA-boxes* is introduced. It is the syntactic equivalent to inconsistent GA-boxes. The definition is by induction on the relation "descendant" which has just been proved noetherian. An GA-box $\mathcal{A}$ occurring in the computation is *contradictory* with respect to a computation iff

- $\mathcal{A}$ does not have descendants and is obviously contradictory, or
- all descendants of $\mathcal{A}$ are contradictory.

Please note that according to this definition $\mathcal{A}_0$ is contradictory iff after the loop in the algorithm all GA-boxes in $\mathcal{M}_r$ are obviously contradictory.

Lemma 3.7.4 (soundness) *An GA-box that is contradictory with respect to a given computation is inconsistent.*

Proof. The proof is by induction on the definition of *contradictory*, with a case analysis according to the transformation rule applied. Assume that a contradictory GA-box $\mathcal{A}$ is given. It has to be shown that it does not have a model.

1) If $\mathcal{A}$ does not have a descendant, it must be obviously contradictory and cannot have a model.

2) For the induction step, assume to the contrary that $\mathcal{A}$ has a model $\mathcal{I}$. It has to be shown that at least one of the descendants of $\mathcal{A}$ has a model. This will be a contradiction to the induction hypothesis, because all descendants of contradictory GA-boxes are contradictory.

This will only be demonstrated for the case of the (R$\forall$) rule. The other cases can be treated similarly.

Assume that the rule has been applied to the assertions $(a, b_1) : v_1, \cdots, (a, b_n) : v_n$, $a : \forall v_1 \cdots v_n.\rho$ generating the descendant $\mathcal{B}$. Please note that $\mathcal{B}$ is a superset of $\mathcal{A}$ and that the only assertion in $\mathcal{B}$ that is not in $\mathcal{A}$ is $(b_1, \cdots, b_n) : \rho$. Hence, it suffices to show that $\mathcal{I}$ satisfies $b : C$—which is an immediate consequence of the definition of the generalized value restriction. □

Lemma 3.7.5 (completeness) *If the initial GA-box A_0 is not contradictory with respect to a given computation, it has a model.*

Proof. If $\mathcal{A}_0$ is not contradictory then there is an GA-box $\mathcal{B} \supseteq \mathcal{A}_0$ in $\mathcal{M}_r$ that is not obviously contradictory. Next an interpretation $\mathcal{I}$ of $\mathcal{B}$ is defined:

1. Because the clash rule related to the concrete domain is not applicable, there is a variable assignment α that satisfies the conjunction of all occurring assertions of the form $P(x_1, \ldots, x_n)$. The interpretation $\mathcal{I}$ interprets all x with $x : \mathcal{D}$ in $\mathcal{B}$ as $\alpha(x)$.

2. The abstract domain $\text{DOM}_\mathcal{I}$ consists of all remaining objects in $\mathcal{B}$.

3. Let ρ be a primitive concept or an abstract predicate. Then $(a_1, \ldots, a_n) \in \rho^\mathcal{I}$ iff $(a_1, \ldots, a_n) : \rho$ occurs in $\mathcal{B}$. The domain rules ensure that all a_i belong to $\text{DOM}_\mathcal{I}$.

4. Let R be a role or attribute. Then $(a, b) \in R^I$ iff $(a, b) : R$ is in $\mathcal{B}$. This is well defined even if R is an attribute, because of the transformation rule (R$\mapsto$), which is not applicable to $\mathcal{B}$. The domain rules ensure that a belongs to $\text{DOM}_\mathcal{I}$.

It is straightforward, but tedious, to show by induction on the size of the assertions that $\mathcal{I}$ is not only an interpretation but also a model of $\mathcal{B}$.

Here only the cases of the generalized value restriction and the negated predicate assertions are demonstrated:

Generalized value restriction: Assume $a : \forall v_1 \cdots v_n.\rho$ is in $\mathcal{B}$. Let any objects $b_1, \ldots, b_n$ be given. If $(a, b_1) \in v_1^{\mathcal{I}}, \cdots, (a, b_n) \in v_n^{\mathcal{I}}$ the transformation rule (R$\forall$) ensures that $(b_1, \ldots, b_n) : \rho$ is in $\mathcal{B}$. By induction hypothesis, $\mathcal{I}$ satisfies this assertion.

Since the b_i were arbitrary, by definition, $\mathcal{I}$ satisfies the generalized value restriction.

Negated abstract predicate assertion: Since $\mathcal{B}$ is not obviously contradictory, the interpretation $\mathcal{I}$ is also model of the negated abstract predicate assertion in $\mathcal{B}$. A similar argument holds for assertions of the form $a : \neg A$ where A is a primitive concept.

Finally, $\mathcal{A}_0 \subseteq \mathcal{B}$ implies that $\mathcal{I}$ is also a model for $\mathcal{A}_0$. □

Chapter 4

Exploring Terminological Knowledge Representation

In this chapter[1] it is explored to which extend the terminological formalism ALCFP($\mathcal{D}$) of the previous chapter may contribute solving the representation and reasoning demands of the application domain introduced in Chapter 2. On the way from simple to more complex features some limitations of the representation and reasoning power of ALCFP($\mathcal{D}$) will be discovered. In the examples, the generic terminological formalism has been instantiated by the concrete domain of real numbers $\mathcal{R}$ to ALCFP($\mathcal{R}$).

Some of the discovered limitations are of principal nature. So this chapter serves as a motivation for the development of the generic rule formalism of the next chapter that takes a first-order logic like ALCFP($\mathcal{D}$) with restricted expressiveness and constructs a more expressive, semidecidable rule formalism.

4.1 Geometric Primitives and Elementary Features

The geometry, as the main ingredient of a CAD drawing, is given as a collection of rotational-symmetric surfaces that are fixed to the symmetry axis of the lathe work. An important geometric element is the truncated cone. Since the surfaces are fixed to an axis, they can be characterized by four real numbers r_1, r_2, c_1, and c_2 (Figure 2.2).

Because not all quadruples correspond to truncated cones, the values of their components have to be restricted: The radii are non-negative and the associated surface should not be degenerated to a line, a circle, or even a point. If these restrictions are represented by the four place predicate truncone-condition over the concrete domain

[1]This chapter is a revised version of paragraphs in [Baader and Hanschke, 1992; Hanschke and Hinkelmann, 1992; Boley *et al.*, 1993].

of real numbers[2] the concept of a *trun*cated *cone* could be defined by

$$\mathsf{truncone} \;=\; \exists(r_1, r_2, c_1, c_2).\mathsf{truncone\text{-}condition}$$

This definition can be specialized to a cylinder by further restricting the radii as being equal using equality on real numbers and the conjunction operator $\sqcap$. Similarly, the definitions of ascending and descending truncated cones, rings, etc. can be obtained by specialization. Truncated cones that are not cylinders are defined as the most specific generalization of ascending and descending truncated cones using the disjunction operator $\sqcup$. An equivalent definition would be $\mathsf{not\text{-}cylinder} = \mathsf{truncone} \sqcap \forall(r_1 \neq_{\mathcal{R}} r_2)$.

$$\begin{array}{lcl}
\mathsf{cylinder} & = & \mathsf{truncone} \sqcap \forall(r_1 =_{\mathcal{R}} r_2) \\
\mathsf{asc\text{-}tc} & = & \mathsf{truncone} \sqcap \forall(r_1 <_{\mathcal{R}} r_2) \\
\mathsf{desc\text{-}tc} & = & \mathsf{truncone} \sqcap \forall(r_1 >_{\mathcal{R}} r_2) \\
\mathsf{ring} & = & \mathsf{truncone} \sqcap \forall(c_1 =_{\mathcal{R}} c_2) \\
\mathsf{asc\text{-}ring} & = & \mathsf{ring} \sqcap \mathsf{asc\text{-}tc} \\
\mathsf{desc\text{-}ring} & = & \mathsf{ring} \sqcap \mathsf{desc\text{-}tc} \\
\mathsf{not\text{-}cylinder} & = & \mathsf{asc\text{-}tc} \sqcup \mathsf{desc\text{-}tc}
\end{array}$$

To improve readability, infix notation has been used for the comparison operators in the value restrictions.

The application also needs concepts that describe more than a single surface. So it is necessary to aggregate the primitive surfaces. For instance, a biconic comprises two neighbored truncated cones (Figure 2.3).

$$\begin{array}{lcl}
\mathsf{biconic} & = & \exists \mathsf{left.truncone} \sqcap \\
 & & \exists \mathsf{right.truncone} \sqcap \\
 & & \forall(\mathsf{left} \circ c_2 =_{\mathcal{R}} \mathsf{right} \circ c_1) \sqcap \\
 & & \forall(\mathsf{left} \circ r_2 =_{\mathcal{R}} \mathsf{right} \circ r_1)
\end{array}$$

Here the attributes left and right play the role of *part-of* attributes linking a biconic to its components. Informally speaking, an object is a member of $\exists$left.truncone iff it has a truncated cone as a filler for left. The expression $\forall(\mathsf{left} \circ c_2 =_{\mathcal{R}} \mathsf{right} \circ c_1)$ forces the right center of the left truncated cone to be equal to the left center of the right truncated cone. This role interaction of parameters has been represented using the equality predicate $=_{\mathcal{R}}$ of the concrete domain. An alternative definition of biconic is

$$\begin{array}{l}
\exists \mathsf{left.truncone} \sqcap \exists \mathsf{right.truncone} \sqcap \\
\forall \mathsf{left, right.neighbored}
\end{array}$$

[2]This predicated of the concrete domain $\mathcal{R}$ could be defined by

$$\begin{array}{lll}
\mathsf{truncone\text{-}condition}(r_1, r_2, c_1, c_2) :\Leftrightarrow & r_1 \geq_{\mathcal{R}} 0 \wedge r_2 \geq_{\mathcal{R}} 0 \wedge & \\
 & (c_1 =_{\mathcal{R}} c_2 \wedge r_1 \neq_{\mathcal{R}} r_2 \vee & \\
 & \; c_1 \neq_{\mathcal{R}} c_2 \wedge (r_1 >_{\mathcal{R}} 0 \vee r_2 >_{\mathcal{R}} 0)). &
\end{array}$$

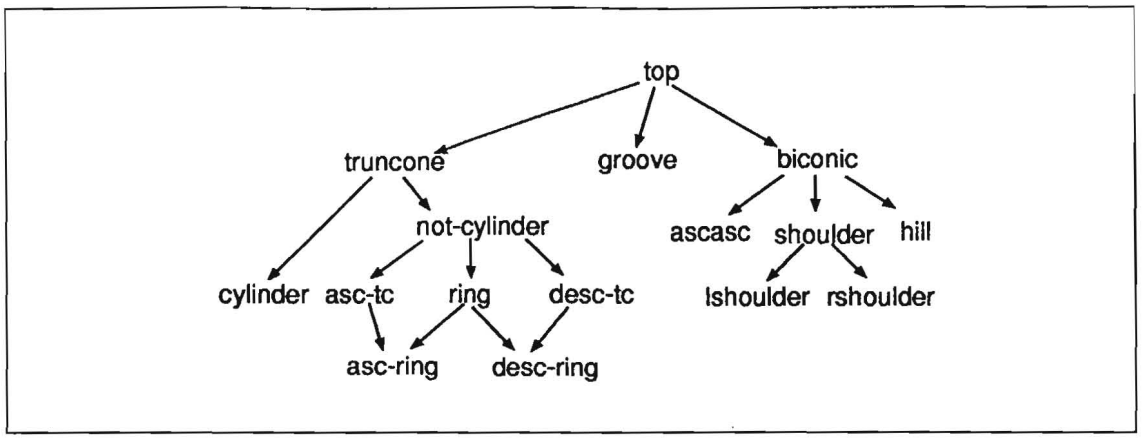

Figure 4.1: The Subsumption Graph of the Sample Terminology

where neighbored is a binary, abstract predicate representing the interaction of left and right. This approach can be more sensible if the CAD model provides an explicit topological model expressing neighborhood on an abstract level.

However, specializations of biconic can be defined using the value restriction operator $\forall$. Informally speaking, an object belongs to $\forall$left.cylinder if it does not have any attribute filler or it has a cylinder as attribute filler for left.

$$\begin{array}{lcl} \text{ascasc} & = & \text{biconic} \sqcap \forall\text{left.asc-tc} \sqcap \forall\text{right.asc-tc} \\ \text{hill} & = & \text{biconic} \sqcap \forall\text{left.asc-tc} \sqcap \forall\text{right.desc-tc} \\ \text{rshoulder} & = & \text{biconic} \sqcap \forall\text{left.cylinder} \sqcap \forall\text{right.asc-ring} \\ \text{lshoulder} & = & \text{biconic} \sqcap \forall\text{right.cylinder} \sqcap \forall\text{left.desc-ring} \\ \text{shoulder} & = & \text{lshoulder} \sqcup \text{rshoulder} \end{array}$$

The next concept shows how two shoulders can be combined to a groove.

$$\begin{array}{lcl} \text{groove} & = & \exists\text{left.lshoulder} \sqcap \exists\text{right.rshoulder} \sqcap \\ & & \forall(\text{left} \circ \text{right} = \text{right} \circ \text{left}) \end{array}$$

Here the relation between the components (i.e., shoulders) of the groove have been modeled by an universal (attribute) agreement.[3] The *concept classification* service arranges the concepts as shown in Figure 4.1.

To represent a particular lathe work in a terminological system, the assertional formalism, called A-box, is employed. It allows to instantiate the concepts with instances and to fill in their attributes. A single truncated cone could for example be represented by the following A-box:

$$\begin{array}{lll} (\text{tc}_1) : \text{truncone}, & & \\ (\text{tc}_1, 0) : \text{c}_1, & (\text{tc}_1, 10) : \text{r}_1, & \\ (\text{tc}_1, 5) : \text{c}_2, & (\text{tc}_1, 10) : \text{r}_2 & \end{array} \tag{4.1}$$

[3]Note that in $\forall(\text{left} \circ \text{right} = \text{right} \circ \text{left})$ the symbol $=$ denotes 'global' equality defined as $\{(x, x) | \; x \in \text{DOM}_{\mathcal{I}} \cup \text{DOM}_{\mathcal{R}}\}$, which is not the equality $=_{\mathcal{R}}$ of the concrete domain.

Strictly speaking, attribute-filler assertions with concrete objects (like 10) are not allowed. But, an assertion like $(\mathsf{tc}_1, 10) : \mathsf{r}_1$ can be seen as an abbreviation of two assertions $(\mathsf{tc}_1, a) : \mathsf{r}_1$ and $p_{10}(a)$ where a is a fresh object and p_{10} is a unary predicate from the concrete domain with the extension $\{10\}$. The *object classification service* of the A-box computes the realization $\{\mathsf{cylinder}\}$ of tc_1.

4.2 Some Limitations

Terminological formalisms focus unary (concepts) and binary predicates (roles), and, furthermore, the structure of the formulas in which these predicates may occur is rather restricted. As an achievement of the careful design of $\mathsf{ALCFP}(\mathcal{D})$ the reasoning problems associated with the inference services are decidable (if $\mathcal{D}$ is an admissible concrete domain) and the formalism is still expressive enough to serve some needs of realistic applications.

Note, that terminological knowledge is represented independent of its intended use, it is not necessary for a knowledge engineer to consider restrictions of the operational semantics. In particular, there does not exist a notion of left-to-right, top-down, or bottom-up evaluation of a terminological knowledge base or query as it is common with rule formalisms.

Complementary to these advantages there are some limitations with respect to expressive power. The particular limitations illustrated in the following subsections concern representation as well as reasoning.

4.2.1 Aggregation

Terminological reasoning systems directly support the abstraction mechanisms generalization and classification. But they do not bother about aggregation. For instance, consider a truncated cone tc_2 that neighbors the cylinder tc_1 introduced in (4.1):

$$\begin{array}{lll} (\mathsf{tc}_2) : \mathsf{truncone}, & & \\ (\mathsf{tc}_2, 5) : \mathsf{c}_1, & (\mathsf{tc}_2, 10) : \mathsf{r}_1, & \\ (\mathsf{tc}_2, 5) : \mathsf{c}_2, & (\mathsf{tc}_2, 15) : \mathsf{r}_2 & \end{array} \tag{4.2}$$

The object classification service would derive that tc_2 is an ascending ring. But it **cannot** detect that tc_1 and tc_2 together form a 'biconic'—unless the objects are aggregated to a **single** instance. Once there is an object bi with assertions

$$(\mathsf{bi}, \mathsf{tc}_1) : \mathsf{left}, \quad (\mathsf{bi}, \mathsf{tc}_2) : \mathsf{right} \tag{4.3}$$

bi can be classified as a rshoulder.

But this kind of introduction of new instances is not a standard operation in terminological reasoning systems. The selection of instances that are composed to a new object does not depend on terminological knowledge. On the contrary, knowledge about aggregation of instances is part of the assertional box. This can easily be seen

in the case that the aggregation is not unique. To illustrate this, let us consider a simple configuration example. [4] Let a terminal be defined as a keyboard connected to a screen. Suppose there are two keyboards k_1 and k_2 and two screens s_1 and s_2. If and how screens and keyboards are put together is not part of the terminological but of the assertional component. So there must be a rule which describes under which particular circumstances (for example because of customer requirements) k_1 and s_2 are connected to form a terminal t_1.

Hence, there is a representation deficit (i.e., it cannot be expressed when a aggregation has to take place) and a reasoning deficit (i.e., none of the terminological inference services aggregates objects to new objects).

4.2.2 Derived Attribute and Role Fillers

There is a further difficulty associated with assertional reasoning in terminological systems. Consider the following concept definitions for regular, tall, and flat shoulders:

$$\begin{aligned}
\mathsf{rshoulder\text{-}with\text{-}hw} &= \mathsf{rshoulder} \sqcap \\
&\quad \exists(\mathsf{height} =_{\mathcal{R}} \mathsf{right} \circ r_2 - \mathsf{right} \circ r_1) \sqcap \\
&\quad \exists(\mathsf{width} =_{\mathcal{R}} \mathsf{left} \circ c_2 - \mathsf{left} \circ c_1) \\
\mathsf{regular\text{-}rshoulder} &= \mathsf{rshoulder\text{-}with\text{-}hw} \sqcap \forall(\mathsf{height} =_{\mathcal{R}} \mathsf{width}) \\
\mathsf{tall\text{-}rshoulder} &= \mathsf{rshoulder\text{-}with\text{-}hw} \sqcap \forall(\mathsf{height} >_{\mathcal{R}} \mathsf{width}) \\
\mathsf{flat\text{-}rshoulder} &= \mathsf{rshoulder\text{-}with\text{-}hw} \sqcap \forall(\mathsf{height} <_{\mathcal{R}} \mathsf{width})
\end{aligned}$$

The expression $\exists(\mathsf{height} =_{\mathcal{R}} \mathsf{right} \circ r_2 - \mathsf{right} \circ r_1)$ is a mix-fix notation for the application of a three-place predicate of $\mathcal{R}$ to the attribute chainings height, right $\circ$ r_2, and right $\circ$ r_1 in a generalized exists-in restriction.

The object classification cannot identify the aggregate bi of the above example as a regular shoulder. This is a representational problem, because it cannot be expressed that a height and a width is associated with each shoulder and depends functionally on its radii and centers, and analogue to aggregation, it is a reasoning problem, because there does not exist a service that would automatically introduce the additional attribute fillers.

The problem may occur in the abstraction phase (for example, when the feature regular-rshoulder is not found in the A-box $\mathcal{A}$ comprised of (4.1), (4.2), (4.3)) as well as in the refinement phase (for example, when it does not lead to an inconsistent A-box if bi : flat-rshoulder is added to $\mathcal{A}$).

4.2.3 Sequences

Probably the most sever restriction is that sequences cannot be represented in an adequate manner. For example, in the considered application domain it is important to describe classes of lathes which are sequences of geometric primitives. The problem

[4] This telling example is due to Knut Hinkelmann.

is that these sequences have a finite, but varying and not a priori bounded length. It is quite simple to define concepts for features such as

'1 truncated cone', '2 truncated cones', ...

This can be done as in the following terminology:

$$\begin{array}{lcl} \mathsf{last} & = & \forall \mathsf{tail}.\bot \\ \mathsf{connected} & = & \forall \mathsf{head}, \mathsf{tail} \circ \mathsf{head}.\mathsf{neighbored} \\ \mathsf{one} & = & \exists \mathsf{head}.\mathsf{truncone} \sqcap \mathsf{last} \\ \mathsf{two} & = & \exists \mathsf{head}.\mathsf{truncone} \sqcap \exists \mathsf{tail}.\mathsf{one} \sqcap \mathsf{connected} \\ & \ldots & \end{array}$$

Here $\bot$, as usual, stands for the empty concept.

But it remains the problem to represent the most specific generalization (union) of these *infinitely* many features (concepts). The resulting concept could be termed a 'sequence of neighbored truncated cones'. It should be noted that its specialization 'ascending sequence of truncated cones' (see below in 4.5) is essential for characterizing the production classes of lathes.

Adding Transitive Closure

The formalism ALCF^+ is an extension of ALCF that can satisfy the demands of the problem domain for representing sequences. [5] The basic idea of this extension is to allow role and attribute *terms* in value-restrictions and exists-in restrictions instead of just allowing role and attribute *names* as in ALCF.

Definition 4.2.1 (syntax of ALCF^+) *The* role/attribute terms *are built from role and attribute names with*

union	$(R \sqcup S)$,
composition	$(R \circ S)$, *and*
transitive closure	$(\mathsf{trans}(R))$

of roles and attributes. Concept terms in ALCF^+ *are defined as in* ALCF *with the only difference that role/attribute terms can be used in value-restrictions and exists-in restrictions.*

For example, a sequence of truncated cones can be defined as follows:

$$\begin{array}{lcl} \mathsf{sequence} & = & \exists \mathsf{head}.\mathsf{truncone} \sqcap \\ & & \forall \mathsf{trans}(\mathsf{tail}).\exists \mathsf{head}.\mathsf{truncone} \end{array} \tag{4.4}$$

Since ALCF^+ does not provide concrete domains, truncone is a primitive (i.e., not further defined) concept in this terminology and it is not expressed that the truncated cones are neighbored.

[5] This extension is due to Franz Baader, see for example [Baader, 1991], [Baader and Hanschke, 1992].

Definition 4.2.2 (semantics of ALCF^+) *The interpretation can be extended to attribute/role terms in the obvious way:* $(R \sqcup S)^{\mathcal{I}} = R^{\mathcal{I}} \cup S^{\mathcal{I}}$, $(R \circ S)^{\mathcal{I}} = \{(x,y); \exists z : (x,z) \in R^{\mathcal{I}}$ *and* $(z,y) \in S^{\mathcal{I}}\}$, *and* $\mathsf{trans}(R)^{\mathcal{I}} := \bigcup_{n \geq 1} (R^{\mathcal{I}})^n$.

In [Baader, 1991] it is shown that for ALC^+ (i.e., ALCF^+ *without* attributes) the subsumption problem is decidable. A close look at the algorithm for ALC^+ (which is much too complex to be sketched here) reveals that the result also holds for ALCF^+, that means, for concept terms C, D and a terminology $\mathcal{T}$ over ALCF^+ (with attributes and roles) it is decidable whether C subsumes D.

Combining the Extensions

Up to now $\mathsf{ALCFP}(\mathcal{D})$ and ALCF^+, which are both extensions of ALCF, have been considered separately. Now consider the language $\mathsf{ALCFP}^+(\mathcal{D})$ which is obtained if both extensions are combined.

To represent all relevant knowledge of the application domain, one would like to have the representational facilities of both formalisms available. With $\mathcal{R}$ as the concrete domain this language is expressive enough to define concepts that are of great importance for the application domain, such as a 'sequence of neighbored truncated cones' (seq-tc) and its specialization 'ascending sequence of truncated cones' (aseq-tc):

$$\begin{array}{lcl} \mathsf{seq\text{-}tc} & = & \mathsf{sequence} \sqcap (\mathsf{last} \sqcup \mathsf{connected}) \sqcap \\ & & \forall \mathsf{trans(tail).(last} \sqcup \mathsf{connected)} \\ \mathsf{aseq\text{-}tc} & = & \mathsf{seq\text{-}tc} \sqcap \forall \mathsf{head.asc\text{-}tc} \sqcap \\ & & \forall \mathsf{trans(tail)} \circ \mathsf{head.asc\text{-}tc} \end{array} \tag{4.5}$$

where the other concepts are as above.

The price that has to be paid for this expressiveness is that it cannot be decided in general whether a concept C subsumes a concept D in this language.

This can be shown by reducing the Post Correspondence Problem to the subsumption problem for this language. The reduction uses only very simple predicates from real arithmetic, namely equalities between linear polynomials in at most two variables. The interested reader is referred to [Baader and Hanschke, 1991b; Baader and Hanschke, 1992] where a similar result is proved. [6] The only difference is that $\mathsf{ALCFP}(\mathcal{D})$ is replaced by a less expressive language without abstract predicates and general roles in generalized value and exists-in restrictions and without universal or existential attribute (dis)agreements. An analogue result is obtained if instead of adding transitive closure to $\mathsf{ALCFP}(\mathcal{D})$ cyclic definitions were introduced in terminologies of $\mathsf{ALCFP}(\mathcal{D})$.

In [Hanschke and Würtz, 1993] the undecidability of the satisfiability problem of a very (most simple?) class of logic programs (comprising one fact $P(g_1, \ldots, g_n)$ one clause $P(l_1, \ldots, l_n) \leftarrow P(r_1, \ldots, r_n)$ and one goal $P(f_1, \ldots, f_n)$) is proved. Both this

[6] The proof is due to Franz Baader.

result and the undecidablity for $\mathsf{ALCFP}^+(\mathcal{D})$ suggest that as soon as a varying-size aspect can be represented in a formalism, further extensions have to be made very careful. Otherwise the associated reasoning problems get undecidable.

Chapter 5

An Epistemic Formalism

In the previous chapter it has been demonstrated that terminological formalisms, in general, and the formalism ALCFP($\mathcal{R}$), in particular, are both useful for representing terminological knowledge and limited with respect to their expressive power. The main issue of this chapter is the homogeneous integration of the special-purpose reasoning power of a terminological formalism with the general-purpose representation and reasoning power of a semidecidable (computationally complete) rule formalism. Therefore a declarative generic rule scheme is developed and applied to the terminological formalism ALCFP($\mathcal{R}$). The chapter concludes by showing how the representation and reasoning problems left open in the previous chapter can be dealt with.

5.1 Introduction

The proposed generic rule formalism is based on rules of the form

$$\phi_0 \rightsquigarrow \phi_1 | \ldots | \phi_n \tag{5.1}$$

where $n \geq 0$ and ϕ_i are formulas of a first-order logic satisfying certain requirements and the symbol "$\rightsquigarrow$" stands for a weak form of implication explained later. The formalism is parametrized by the first-order logic which is referred to as the *condition formalism*. For instance, it will be shown that the terminological formalism ALCFP($\mathcal{D}$) of Chapter 3 can be seen as a condition formalism. Term equations (and negated term equations) induce another relevant condition logic (cf. Section 5.5). The operational semantics of the generic rule formalism generalizes the way productions rules are applied to a fact base. If a rule is triggered one of the ϕ_i, $1 \leq i \leq n$ in the head is (don't know) non-deterministically selected and added to the fact base.

Informally, such a rule says "if ϕ_0 is believed, then one of ϕ_1, ϕ_2, ...,ϕ_n is believed." For $n = 0$ the rule is a *denial* saying that whenever ϕ_0 is believed, the current state is inconsistent. For $n = 1$ the rules are very close to production rules. If ϕ_0 is very simple and $n > 1$, the operational semantics of these rules has much in common with SLD resolution (cf. Section 5.5.1).

Hence, this formalism combines deterministic, data-driven, bottom-up reasoning (as required for abstraction) with non-deterministic, goal-directed, top-down search (as required by the association and the refinment phase).

5.1.1 Operational Semantics

The operational semantics can be considered as production rule-like inferences combined with backtracking search. First of all, there is a fact base $\mathcal{A}_0$. Objects occurring in the fact base are substituted for variables in the premise of a rule. Then it is checked whether the fact base entails (defined formally in Definition 5.2.4) the instantiated premise with respect to the condition logic. If not, other objects may be tried with another rule. Otherwise, one of the alternatives in the head of the instantiated rule is added to the fact base. If the fact base gets inconsistent (defined formally in Definition 5.2.4) with respect to the condition logic, backtracking takes place: The computation resumes at the most recent point where another alternative in the head of a rule can be selected. If all instantiations of rules with a premise that is entailed by the current fact base $\mathcal{A}$ have been applied, and if the current fact base $\mathcal{A}$ is consistent, $\mathcal{A}$ is an answer computed by the set of rules for $\mathcal{A}_0$.

Figure 5.1 shows a naive implementation of this operational semantics written in a pseudo programming language.

For a fact base $\mathcal{A}$ and a set of rules Pgm the computation started with the call closure($\mathcal{A}$, Pgm, $\emptyset$, $\emptyset$) enumerates all consistent answers that can be computed for $\mathcal{A}$ by Pgm or the function runs for ever internally generating an infinite consistent fact-base.

The third and the fourth argument are auxiliary parameters used to organize the search for an entailed premise. The pairs in the set entailed-triggers represent all successful rule applications which have led to the current fact base. The pairs in failed-triggers represents all failed attempts to entail an instantiated premise by the current fact base.

The don't-know non-determinism of the selection of an alternative of the head of a rule is explicitly coded in the function do-not-know-apply. The non-deterministic selection of a trigger for a rule in (i) is don't-care and has to be *fair*, i.e., if a certain pair $(r, \phi_0\sigma)$ *could* be selected in a certain stage of the computation, it *is* selected eventually or all subsequent selections of alternatives eventually lead to an inconsistent fact base.

With this fairness property the algorithm cannot loop infinitely in an inconsistent branch of the don't know search space. In contrast, Prolog with its left-to-right goal selection may loop infinitely in a goal which could be detected as being not provable in finite time (with another goal selection strategy).

Note that the function do-not-know-apply implements a depth-first search strategy (as in Prolog). Hence the inference algorithm may run forever (generating internally an infinite consistent fact base) although there may exist a finite consistent answer and (due to the fairness property) the algorithm does not loop infinitely in

```
define closure(fact-base, Pgm, entailed-triggers, failed-triggers) :=
  if consistent(fact-base) then
    (i) Let φ0 ⇝ φ1|...|φn be a renaming with fresh variables of a rule
    r ∈ Pgm and let σ be a mapping from variables of φ0 to objects of
    fact-base such that (r, φ0σ) ∉ entailed-triggers ∪ failed-triggers
    if this is not possible then
      show fact-base
      ; returning false initiates search for next answer
      false
    elseif fact-base entails φ0σ then
      do-not-know-apply(
        λfact.closure(fact-base ∪ {fact}, Pgm, entailed-triggers ∪ {(r, φ0σ)}, ∅),
        {φ1σ, ···, φnσ})
    else
      closure(fact-base, Pgm, entailed-triggers, failed-triggers ∪ {(r, φ0σ)})
    endif
  else
    false
  endif


define do-not-know-apply(f, set) :=
  if set = ∅ then
    false
  else
    (ii) Let φ ∈ set and result := f(φ)
    if result = false
      then
        do-not-know-apply(f, set \ φ)
    else
      result
    endif
  endif
```

Figure 5.1: A Naive Implementation of the Rule Formalism

an *in*consistent branch of the computation. Imposing a left-to-right strategy on the selection of an alternative in the head, would leave some control to the knowledge engineer who writes down the rules. This would be analogous to the rule-selection strategy in Prolog. It is also possible to implement a breadth-first search (for example, by iterative deepening) to avoid this kind of incompleteness.

Note that the generic inference algorithm of the rule scheme just requires the functions consistent and entails to be provided by the condition formalism.

5.1.2 Logical Reading

The rules should not be regarded as logical implications in the classical sense. For example, the operational semantics of the formalism does not take care of contra positions: If there is a rule $\phi \rightsquigarrow \phi'$ and $\neg\phi'$ is believed, it will not derive $\neg\phi$. The operator $\rightsquigarrow$ also differs from classical implication in the following sense: If $\phi \vee \phi'$ holds and there are rules $\phi \rightsquigarrow \phi''$ and $\phi' \rightsquigarrow \phi''$, then ϕ'' is not derived by these rules. Finally, assume that there is a rule $\phi(x) \rightsquigarrow \phi'(x)$ with a variable x, which is implicitly universally quantified. Then the rule is only triggered if there is an object a in the current fact base such that $\phi(a)$ is implied by the fact base. Thus, all variables in the premise of a rule have to be instantiated by objects which occur explicitly in the fact base.

These restrictions enable efficient processing of the rules. The *trigger rules* in [Brachman *et al.*, 1991; Edelmann and Owsnicki, 1986; MacGregor, 1988] have similar restrictions in their operational semantics. A trigger rule $A \rightsquigarrow B'$ can be regarded as a special case of (5.1) where ϕ_0 and ϕ_1 are concepts and $n = 1$. In [Donini *et al.*, 1992] a semantics based on the epistemic operator K, standing for 'knows', is proposed which coincides with the operational semantics. Levesque has introduced the K operator in his ask and tell framework [Levesque, 1984].

In [Lifschitz, 1991] Lifschitz relates minimal believe logics to the semantics of some logic programming formalisms including general, disjunctive logic programs. He replaced the letter K by the letter B reflecting his preference of 'believe' in place of 'knowledge' as the intuition behind his logic.

This idea of a *minimal believe logic* is also the key to the semantics of the rule formalism introduced here. However, non of the mentioned formalizations of an epistemic logic is appropriate as the basis for a model-theoretic semantics of the rules. Compared to [Donini *et al.*, 1992] the formalism considered here offers more complex premises, disjunctions in the conclusions, and variables occurring only in the head. It is also more general, because it tolerates the possible presence of equality "=" and the possible absence of a unique-name assumption.

Premises with more then one variable together with the absence of the unique-name assumption induce a major technical problem. Consider, for instance, a fact base just consisting of the fact $p(x, y)$, which is associated with the epistemic formula $\exists x, y(Bp(x, y))$, and a program consisting of a rule $p(x, x) \rightsquigarrow q$, which is associated with $\forall x(Bp(x, x) \Rightarrow q)$. Note, that the rule cannot be applied to the fact base. The

soundness result below (Proposition 5.4.1) implies that each epistemic model of (the computed answer) $\exists x, y(Bp(x,y))$ satisfies the program and the fact base.

What is an epistemic model? Roughly, the formalizations of epistemic logics in [Reiter, 1990; Lifschitz, 1991; Levesque, 1984; Donini *et al.*, 1992] all have the same structure. The following definitions can be seen as a simplification of the logic presented in [Lifschitz, 1991] where an additional modal operator *not* is considered. An epistemic interpretation $(\mathcal{I}, \mathcal{M})$, which is also referred to as a *structure*, consists of an interpretation $\mathcal{I}$ of an underlying first-order logic and a set $\mathcal{M}$ of such interpretations where all interpretations $\mathcal{J} \in \mathcal{M}$ and $\mathcal{I}$ share the same domain, D say. For the parameters $d \in D$ names $n_1, n_2, \cdots$ are introduced. These names are in a one-to-one correspondence to the parameters in D. The notion of satisfiability for structures is inductively defined as follows. If a structure $(\mathcal{I}, \mathcal{M})$ satisfies an epistemic formula ϕ, this is written as $\mathcal{M}, \mathcal{I} \models \phi$.

1. $\mathcal{M}, \mathcal{I} \models \phi$:iff $\mathcal{I} \models \phi$, for a closed first-order formula ϕ.
2. $\mathcal{M}, \mathcal{I} \models \phi \wedge \phi'$:iff $\mathcal{M}, \mathcal{I} \models \phi$ and $\mathcal{M}, \mathcal{I} \models \phi'$.
3. $\mathcal{M}, \mathcal{I} \models \exists x(\phi(x))$:iff there exists a name n such that $\mathcal{M}, \mathcal{I} \models \phi(n)$.
4. $\mathcal{M}, \mathcal{I} \models \neg\phi$:iff not $\mathcal{M}, \mathcal{I} \models \phi$.
5. $\mathcal{M}, \mathcal{I} \models B\phi$:iff $\mathcal{J}, \mathcal{M} \models \phi$, for all $\mathcal{J} \in \mathcal{M}$.

Then an *epistemic model* $(\mathcal{I}, \mathcal{M})$ of ϕ is a structure with $\mathcal{M}, \mathcal{I} \models \phi$ that is maximal with respect to $\leq$. Here $\leq$ is defined by $(\mathcal{I}, \mathcal{M}) \leq (\mathcal{I}', \mathcal{M}')$:iff $\mathcal{M} \subseteq \mathcal{M}'$. [1]

In the example, let $(\mathcal{I}, \mathcal{M})$ be a model of $\exists x, y(Bp(x,y))$. According to the definition there exist names n_1, n_2 such that $\mathcal{J} \models p(n_1, n_2)$, for all $\mathcal{J} \in \mathcal{M}$. Please observe that both sets of interpretations, defined below, induce epistemic models of $\exists x, y(Bp(x,y))$.

1. $\mathcal{M}_= := \{\mathcal{I} |\ \mathcal{I}$ is an interpretation over D and $(d,d) \in p^{\mathcal{I}}\}$, for some $d \in D$.
2. $\mathcal{M}_{\neq} := \{\mathcal{I} |\ \mathcal{I}$ is an interpretation over D and $(d_1, d_2) \in p^{\mathcal{I}}\}$, for some $d_1, d_2 \in D$ with $d_1 \neq d_2$.

Obviously, neither $\mathcal{M}_= \subseteq \mathcal{M}_{\neq}$ nor $\mathcal{M}_{\neq} \subseteq \mathcal{M}_=$, and $(\mathcal{I}, \mathcal{M}_=)$ is not an epistemic model of the rule $\forall x(Bp(x,x) \Rightarrow q)$—contrary to the desired soundness result.

The example suggests that the problem is related to an interplay of the modal operator and the quantifiers. There happens something interesting if the scope of two existential quantifications interact with the scope of an occurrence of the modal operator.

Let $(\mathcal{I}, \mathcal{M})$ be a structure. For $B(p \vee q)$ all ways to make $p \vee q$ true may be covered by $\mathcal{M}$. Similarly, $\mathcal{M} := \{\mathcal{I} |\ \mathcal{I}$ is an interpretation over D and $\exists x, y(p(x,y))$

[1] The definitions of the other approaches vary in the treatment of $\mathcal{I}$ and $\mathcal{I}'$.

is true in $\mathcal{I}\}$ covers all possible ways to satisfy $\exists x, y(p(x,y))$ with interpretations over a fixed domain D. But consider $\exists x, y(Bp(x,y))$. It is impossible that $\mathcal{M}$ covers all possibilities how $p(x,y)$ can be made true given that there are objects d_1 and d_2 for which it is *just*[2] required that (d_1, d_2) is in the extension of p. The set $\mathcal{M}$ must be incomplete in this respect, because selecting names n_1 and n_2 either with $n_1 = n_2$ or $n_1 \neq n_2$ to substitute for x and y is a commitment to either a set of type $\mathcal{M}_=$ or $\mathcal{M}_{\neq}$, respectively. Note that this problem does not occur with trigger rules.

In the following section epistemic logics are formalized using *partitions with infinite equivalence classes as interpretation domains and the notion of pre variable assignments.* This conception enables an epistemic model to vary also over all possible variable assignments by assigning the elements of the range of a pre assignment to different equivalence classes in different interpretations. The key result is Proposition 5.2.7.

5.2 Condition Formalisms and Minimal Belief

The semantics of an expression $B\phi(x)$ is usually defined by an intersection of the extensions of ϕ in all possible interpretations, or, as above, by the use of names. Equivalently, an assignment for the variables in $\phi(x)$ could be fixed, before the truth values of $\phi(x)$ with respect to to this assignment in all possible interpretations are conjoined.

An important requirement, stemming from the operational semantics is that $p(c)$ implies $\exists x(Bp(x))$. An epistemic model satisfies the latter if their is an assignment α to x such that in all interpretations of the epistemic model $p(x\alpha)$ is true. In [Donini *et al.*, 1992] this problem is solved by fixing the domain and the assignments of constants to elements of the domain for all interpretations. This was possible since they do have a unique-name assumption: without loosing generality all constants can be mapped to pairwise distinct parameters of the domain in the same manner in all interpretations. But if the logic does not have a unique-name assumption, it should be possible for two constants to be identified in one interpretation and to be different in another.

Here a different approach is taken. The object c is considered as a variable that occurs in the scope of $\exists$ and B and the formula $\exists y(Bp(y))$ replaces $p(c)$. As a by-product, the names of objects introduced in a computation are irrelevant (as long as they are 'fresh'). If the objects were constants it would have been more complicated to formulate the completeness result (Proposition 5.4.2), because in a computation new constants may be generated. The names of these constants are irrelevant for epistemic models of the program but would restrict epistemic models of the computed answers.

The following definition formally defines the notion of a 'condition formalism' which up to this point has only been used in an intuitive sense.

[2]In particular, there is nothing said about $x = y$ or $x \neq y$.

Definition 5.2.1 (condition formalism) *A first-order language* CF *gets a* condition formalism, *if the following notational conventions are adopted and* CF *satisfies the requirements formulated below.*

The domain of an interpretation $\mathcal{I}$ of CF *is denoted by* $\mathrm{DOM}_{\mathcal{I}}$. *A variable assignment is a partial mapping from variables to elements of the domain. An interpretation function $.^{\mathcal{I}}$ assigns a set $\phi^{\mathcal{I}}$ of variable assignments to a formula ϕ such that α is defined on x iff the variable x occurs free in ϕ, for all $\alpha \in \phi^{\mathcal{I}}$. An interpretation $\mathcal{I}$ and a variable assignment α* satisfy *a formula ϕ iff α restricted to the free variables in ϕ is in $\phi^{\mathcal{I}}$. This is written as $\mathcal{I}, \alpha \models \phi$.*

The first-order logic also has to satisfy the following requirements:

- *Condition formalisms are closed under conjunction: Let ϕ and ϕ' be two formulas with sets of variables V and V', respectively. Then $\phi \wedge \phi'$ is also a formula of* CF, *with variables $V \cup V'$, and*

$$(\phi \wedge \phi')^{\mathcal{I}} = \left\{ x \mapsto \left\{ \begin{array}{ll} x\alpha, & \text{if } x \in V \\ x\alpha_1, & \text{if } x \in V' \end{array} \;\middle|\; \begin{array}{l} \alpha \in \phi^{\mathcal{I}}, \alpha_1 \in \phi'^{\mathcal{I}}, \\ \text{and } x\alpha = x\alpha_1, \\ \text{for all } x \in V \cap V' \end{array} \right\} \right.$$

  for each interpretation $\mathcal{I}$. By abuse of notation, sometimes a condition formula is considered as the set of its conjuncts and vice versa.

- *The interpretations are closed under renaming: Let $\mathcal{I}$ be an interpretation with domain $\mathrm{DOM}_{\mathcal{I}}$ and $\pi : \mathrm{DOM}_{\mathcal{I}} \to D$ be some bijection into another set. Then $\mathcal{I}\pi$ is also an interpretation with $\mathrm{DOM}_{\mathcal{I}\pi} = D$ and $\phi^{\mathcal{I}\pi} := \{\alpha\pi |\ \alpha \in \phi^{\mathcal{I}}\}$.*

□

The operational semantics as well as the fixpoint semantics (see below) of the rule formalism may induce potentially infinite sets of formulas. Hence, 'conjunction' is extended to the countable, infinite case. If variable assignments are identified with their graphs, $(\phi \wedge \phi')^{\mathcal{I}}$ could be written as $\{\alpha \cup \alpha' | \alpha \in \phi^{\mathcal{I}}, \alpha' \in \phi'^{\mathcal{I}}$, and $\alpha \cup \alpha'$ is functional$\}$. For a (possibly) infinite set Φ of formulas, conjunction is defined by

$$(\bigwedge \Phi)^{\mathcal{I}} := \left\{ \beta |\ \beta = \cup\{\xi(\phi) |\ \phi \in \Phi\}, \xi : \phi \mapsto \alpha \in \phi^{\mathcal{I}}, \beta \text{ is functional} \right\}$$

The epistemic logic CF(B) defined next is parametrized by a condition formalism CF.

Definition 5.2.2 (syntax of epistemic logic CF(B)) *The syntax of the epistemic logic* CF(B) *is inductively defined as follows:*

- *Every formula of* CF *is an epistemic formula.*

- *If ϕ is an epistemic formula and Φ is a set of epistemic formulas, $\neg\phi$, $\bigwedge\Phi$, $\exists x(\phi)$, and $B\phi$ are epistemic formulas. Here x is a finite or countable infinite tuple of pairwise different variables.* □

The logical connectives $\vee$, $\Rightarrow$, and the quantifier $\forall$ are used in the usual way to abbreviate formulas. An epistemic formula not containing the modal operator B is called *objective*. The mapping $\mathcal{V}$: ⟨epistemic-formulas⟩ $\longrightarrow \mathcal{V}$ retrieves the set of free variables of ϕ.

The semantics of the epistemic logic is defined via sets of interpretations of a special structure. An *admissible interpretation domain* $\text{DOM}_\mathcal{I}$ over D is a non-empty partioning $D_{/\mathcal{I}}$ of a set D where each equivalence class of a partion comprises infinitely many elements of D. By $\Gamma(D)$ the set of all interpretations with an admissible interpretation domain over D is denoted. A *pre variable assignment* α is an injective[3] partial mapping $\alpha : \mathcal{V} \to D$ from variables to elements of D. Note that for $\mathcal{I} \in \Gamma(D)$ a pre variable assignment $\alpha : \mathcal{V} \to D$ induces a variable assignment

$$\alpha^\mathcal{I} : \begin{cases} \mathcal{V} & \to & D_{/\mathcal{I}} \\ x & \mapsto & [x\alpha]_\mathcal{I} \end{cases}$$

Conversely, for each variable assignment $\alpha : \mathcal{V} \to D_{/\mathcal{I}}$ there exists a pre variable assignment β such that $\alpha = \beta^\mathcal{I}$. It follows from the Skolem-Löwenheim Theorem that it is not a restriction to consider interpretations with an admissible interpretation domain over D provided that D is large enough. The cardinality of D should be at least the maximum of ω and the cardinality of the signature of the condition formalism.

Definition 5.2.3 (semantics of epistemic logic CF(B)) *Let $\mathcal{M} \subseteq \Gamma(D)$ be a set of interpretations for some D, $\mathcal{I} \in \Gamma(D)$ an interpretation, and $\alpha : \mathcal{V}_\alpha \to D$ a pre variable assignment. Then $\mathcal{M}$ is an* epistemic interpretation *and the four place (meta-) predicate $\models$ is inductively defined according to the structure of epistemic formulas as follows:*

- *Let ϕ be an objective epistemic formula. Then $\mathcal{M},\mathcal{I},\alpha \models \phi$:iff $\mathcal{I},\alpha^\mathcal{I} \models \phi^\mathcal{I}$.*
- *Let ϕ be an epistemic formula, Φ a finite or countable infinite set of epistemic formulas, x a finite or countable tuple of pairwise different variables, and $\mathcal{V}(x)$ the set of variables in x. Then $\models$ is defined by*

  1. *$\mathcal{M},\mathcal{I},\alpha \models \neg\phi$:iff not $\mathcal{M},\mathcal{I},\alpha \models \phi$*
  2. *$\mathcal{M},\mathcal{I},\alpha \models \bigwedge\Phi$:iff $\mathcal{M},\mathcal{I},\alpha \models \phi$, for all $\phi \in \Phi$*
  3. *$\mathcal{M},\mathcal{I},\alpha \models B\phi$:iff $\mathcal{M},\mathcal{J},\alpha \models \phi$, for all $\mathcal{J} \in \mathcal{M}$*

[3] A mapping ξ is *injective* :iff $x \neq y$ implies $x\xi \neq y\xi$, for all elements x, y of the domain of ξ.

4. $\mathcal{M},\mathcal{I},\alpha \models \exists x(\phi)$ *:iff there is a pre assignment* $\beta : \mathcal{V}_\beta \rightarrow D$ *and a bijection* $\pi : \mathcal{V}(x) \rightarrow \mathcal{V}_\beta$ *such that* $\mathcal{M},\mathcal{I},\alpha \cup \beta \models \phi\pi$ *and* $\mathcal{V}_\alpha \cap \mathcal{V}_\beta = \emptyset$

Similar to condition formulas, a set of epistemic formulas Φ *is identified with a conjunction* $\bigwedge\Phi$. *The triple* $\mathcal{M},\mathcal{I},\alpha$ satisfies ϕ *:iff* $\mathcal{M},\mathcal{I},\alpha \models \phi$. *An epistemic interpretation* $\mathcal{M}$ *satisfies an epistemic formula* ϕ *:iff* $\mathcal{M},\mathcal{I},\alpha \models \phi$, *for all* $\mathcal{I} \in \mathcal{M}$ *and all variable assignments* α. *An* epistemic model $\mathcal{M}$ of an epistemic formula ϕ *is a maximal epistemic interpretation* $\mathcal{M} \subseteq \Gamma(D)$ *satisfying* ϕ. *I.e., if there is an epistemic interpretation* $\mathcal{M}' \subseteq \Gamma(D)$ *satisfying* ϕ *and* $\mathcal{M}' \supseteq \mathcal{M}$, *then* $\mathcal{M}' = \mathcal{M}$. *A closed epistemic formula is* epistemically consistent *:iff it has an epistemic model.* □

A *fact base* $\mathcal{A}_0$ is a set of condition formulas, which may also be considered as a possibly countable infinite conjunction. By abuse of notation, a fact base may also be considered as an epistemic formula: An *epistemic model (resp, interpretation)* $\mathcal{M}$ *of a fact base* $\mathcal{A}$ (infinite or not) is an epistemic model (resp, interpretation) of $\exists x(B\mathcal{A})$ where x is a (possible infinite) tuple comprising exactly the variables that occur free in $\mathcal{A}$.

Definition 5.2.4 (entailment and consistency of CF formulas) *Let* ϕ *and* ϕ' *be two formulas of* CF. *Then* ϕ entails ϕ' *:iff* $\mathcal{I},\alpha \models \phi$ *implies* $\mathcal{I}\alpha \models \phi'$, *for all interpretations* $\mathcal{I}$ *and all variable assignments* α. *A formula* ϕ *is* consistent *:iff there exists* $\mathcal{I}$ *and* α *such that* $\mathcal{I},\alpha \models \phi$. □

Please, recall that the algorithm in Figure 5.1 takes an *entailment test* and a *consistency test* of CF to implement the operational semantics of the rule formalism.

Since fact bases are identified with possibly infinite conjunctions, Definition 5.2.4 carries over to fact bases. Another property of CF, which is always satisfied by a first-order logic, is compactness.

Definition 5.2.5 (compactness) *A condition formalism is* compact *:iff for each set of formulas* Φ *that entails a finite formula* ϕ *there is a finite set* $\Phi' \subseteq \Phi$ *such that* Φ' *entails* ϕ *and all elements of* Φ' *are finite.* □

The premises of the rule formalism shall filter the objects present in the fact base. In particular, they should not be over general (accept any object) and they should not invent new objects not present in the fact base. The following definition, introduces the notion of *filtering condition formulas* which makes the above idea precise.

Definition 5.2.6 (filtering) *An objective formula* ϕ *is* filtering with respect to a variable $x \in \mathcal{V}(\phi)$ and a fact base $\mathcal{A}$ *:iff there exists an* $\mathcal{I}$ *and* α *such that* $\mathcal{I},\alpha \models \mathcal{A}$ *and* $\text{DOM}_\mathcal{I} \neq x\phi^\mathcal{I} := \{x\beta |\ \beta \in \phi^\mathcal{I}\}$. *The objective formula* ϕ *is* filtering *:iff it is filtering with respect to all variables in* $\mathcal{V}(\phi)$ *and all fact bases.*

For example, the premise of $\top(x) \rightsquigarrow p(x)$ is not filtering for any consistent fact base if $\top(x)$ is a unary predicate always interpretated as the whole domain. If such a rule would be allowed, it would be necessary to represent universally quantified formulas in the fact base to get complete inferences. In the example, this would be $\forall x(Bp(x))$. But, then a more general entailment and consistency test would be needed.[4]

The following proposition establishes an important interrelation of the epistemic logic $\mathsf{CF}(B)$ and the underlying condition formalism CF: Roughly, an epistemic model of a fact base comprises all 'characteristic' interpretations.

Proposition 5.2.7 *Let a fact base $\mathcal{A}$, an epistemic model $\mathcal{M} \subseteq \Gamma(D)$ of $\mathcal{A}$, a filtering objective formula ϕ, and a pre assignment $\beta : \mathcal{V}(\phi) \to D$ be given such that $\mathcal{I}, \beta^{\mathcal{I}} \models \phi$, for all $\mathcal{I} \in \mathcal{M}$, and $\mathcal{V}(\mathcal{A})$ and $\mathcal{V}(\phi)$ are disjoint. Then there exists $\sigma : \mathcal{V}(\phi) \to \mathcal{V}(\mathcal{A})$ such that $\mathcal{A}$ entails $\phi\sigma$.*

Furthermore, there exists a pre assignment $\alpha : \mathcal{V}(\mathcal{A}) \to D$ such that $\mathcal{I}, \alpha^{\mathcal{I}} \models \mathcal{A}$, for all $\mathcal{I} \in \mathcal{M}$, and σ can be chosen such that $x\sigma = x\beta\alpha^{-1}$, for all $x \in \mathcal{V}(\phi)$.

Proof. For a mapping ξ denote by $\mathrm{ran}(\xi)$ the set $\{x\xi |\ \xi \text{ is defined on } x\}$. According to the assumptions there exists a pre variable assignment $\alpha : \mathcal{V}(\mathcal{A}) \to D$ such that $\mathcal{I}, \alpha^{\mathcal{I}} \models \mathcal{A}$, for all $\mathcal{I} \in \mathcal{M}$.

Claim 1: $\mathrm{ran}(\alpha) \supseteq \mathrm{ran}(\beta)$

> *Proof.* 1) If $ran(\alpha) = D$ the claim trivialy holds.
>
> 2) Otherwise, assume that the claim does not hold. Then there exists $y \in \mathcal{V}(\phi)$ such that $y\beta \notin \mathrm{ran}(\alpha)$. Since ϕ is filtering, there exists an interpretation $\mathcal{I}_1$ and a variable assignment α_1 such that $\mathcal{I}_1, \alpha_1 \models \mathcal{A}$ and $y(\phi^{\mathcal{I}_1}) \neq \mathrm{DOM}_{\mathcal{I}_1}$.
>
> According to the remarks on admissible interpretation domains and the size of D before Definition 5.2.3 there exists also an interpretation $\mathcal{I}_2$ with an admissible interpretation domain $\mathrm{DOM}_{\mathcal{I}_2}$ such that $\mathcal{I}_2, \alpha_2 \models \mathcal{A}$ and $y(\phi^{\mathcal{I}_2}) \neq \mathrm{DOM}_{\mathcal{I}_2}$. Consequently, there exists an $e \in D$ such that $[e]_{\mathcal{I}_2} \in \mathrm{DOM}_{\mathcal{I}_2} \setminus y(\phi^{\mathcal{I}_2})$. Since the elements of D are not important for $\mathcal{I}_2$ as a first-order logic, these element can be assigned freely to the equivalence classes in $\mathrm{DOM}_{\mathcal{I}_2} = D_{/\mathcal{I}_2}$ as long as the domain of $\mathcal{I}_2$ remains admissible. Thus, it can be assumed without loosing generality that $y\beta \notin \mathrm{ran}(\alpha_2)$ and $y\beta = e$.
>
> Because, α and α_2 are injective, there exists a bijection $\pi : D \to D$ such that $x\alpha = x\alpha_2\pi$, for all $x \in \mathcal{V}(\mathcal{A})$, and $e\pi = e$. Let $\mathcal{J} := \mathcal{I}_2\pi$ be the interpretation in $\Gamma(D)$ that is induced by

[4] For equations over terms of uninterpreted function symbols this would not be a problem. For example, in order to search for a generic representation of all substitutions σ such that the universal closure of $p(t)$ entails $p(s\sigma)$ one would compute the most general (rational) unifier of $t = s$.

- $\text{DOM}_{\mathcal{J}} := \{p\pi |\ p \in \text{DOM}_{\mathcal{I}_2}\}$[5] and
- $\psi^{\mathcal{J}} := \psi^{\mathcal{I}_2}\pi := \{\gamma\pi |\ \gamma \in \psi^{\mathcal{I}_2}\}$.

By construction $\mathcal{J}, \alpha^{\mathcal{J}} \models \mathcal{A}$ and $[e\pi]_{\mathcal{J}} \notin y(\phi^{\mathcal{J}})$. Hence $\mathcal{J} \in \mathcal{M}$ (because $\mathcal{M}$ is an epistemic model of $\mathcal{A}$) and $\mathcal{J}, \beta^{\mathcal{J}} \not\models \phi$ (because $y\beta = e$). Contradiction. This completes the proof of Claim 1.

Next σ is defined by $x\sigma = x\beta\alpha^{-1}$. This is well defined since pre assignments are injective. It remains to show that $\mathcal{A}$ entails $\phi\sigma$.

Assume not. Then there exists $\mathcal{I} \in \Gamma(D)$ and a pre assignment γ such that $\mathcal{I}, \gamma^{\mathcal{I}} \models \mathcal{A}$ and $\mathcal{I}, \gamma^{\mathcal{I}} \not\models \phi\sigma$. Analogue to α, α_2, the pair α, γ induces a bijection π and an interpretation $\mathcal{I}\pi \in \Gamma(D)$. Again, it turns out that $\mathcal{I}\pi \in \mathcal{M}$ and, according to the assumption, $\mathcal{I}\pi, \beta^{\mathcal{I}\pi} \models \phi$. Using the definition of σ the latter implies $\mathcal{I}\pi, \alpha^{\mathcal{I}\pi} \models \phi\sigma$. With $\alpha = \gamma\pi$, one gets $\mathcal{I}\pi, \gamma\pi^{\mathcal{I}\pi} \models \phi\sigma$, and, finally, $\mathcal{I}, \gamma^{\mathcal{I}} \models \phi\sigma$. Contradiction. □

Last but not least, in this section it is observed that epistemically consistent fact bases always have an epistemic model.

Observation 5.2.8 *Let $\mathcal{A}$ be a fact base, and let $\mathcal{M} \subseteq \Gamma(D)$ be an epistemic interpretation satisfying $\mathcal{A}$. Then by definition there exists a pre assignment α such that $\mathcal{I}, \alpha^{\mathcal{I}} \models \mathcal{A}$, for all $\mathcal{I} \in \mathcal{M}$. Then $\mathcal{M}' = \{\mathcal{I} \in \Gamma(D) |\ \mathcal{I}, \alpha^{\mathcal{I}} \models \mathcal{A}\}$ is an epistemic model of $\mathcal{A}$ and $\mathcal{M} \subseteq \mathcal{M}' \subseteq \Gamma(D)$.* □

5.3 The Epistemic Rule Formalism

This section formally defines the rule scheme which takes a condition formalism CF and constructs an epistemic rule formalism ER(CF). This rule formalism has an epistemic, model-theoretic semantics which is based on the epistemic logic CF(B) of the previous section. It also has a straight forward fixpoint semantics which is sound nd complete with respect to the epistemic semantics. Given an entailment and an consistency test for CF the fixpoint semantics can be implemented on a computer to obtain an inference engine for the rule formalism. Section 5.1.1 has sketched such an implementation.

The next definition formally defines the syntax of the rules and defines a mapping into CF(B).

Definition 5.3.1 (ER(CF)) *Let a condition formalism* CF *with an entailment test and a consistency test be given, and let $\phi_0, \cdots, \phi_n$, $n > 0$, be $n+1$ finite condition formulas. Then by definition*

$$\phi_0 \rightsquigarrow \phi_1 | \cdots | \phi_n$$

[5]Here $p\pi$ is defined as $\{d\pi |\ d \in p\}$.

is a (program) rule *in* ER(CF) *if* ϕ_0 *is filtering. It is identified with the closed epistemic formula*

$$\forall x_0(B\phi_0 \Rightarrow \bigvee_{i=1\cdots n} \exists x_i(B\phi_i))$$

where x_0 *is the tuple of the variables in* $\mathcal{V}(\phi_0)$ *and* x_i, $1 \leq i \leq n$, *are the tuples of variables in* $\mathcal{V}(\phi_i) \setminus \mathcal{V}(\phi_0)$. *For* $n = 0$ *the formula* $\bigvee_{i=1\cdots n} \exists x_i(B\phi_i)$ *is identified with* $\bot$. *The symbol* $\bot$ *denotes a formula of* CF *that is inconsistent.*

A program Pgm *is a finite set of rules.* □

The operational semantics (described in Section 5.1.1) searches for a consistent answer using backtracking. In the fixpoint semantics described her this nondeterminism is handled with a selection function that acts as an oracle. The selection function says which alternative to select in an instantiated rule. The fixpoint operator defined below is parametrized by such a selection function.

Definition 5.3.2 (fixpoint semantics of ER(CF)**)** *Let a program* Pgm *be given. A* selection function

$$\mathsf{sel} : (r, \phi_0\sigma) \longmapsto k$$

assigns to a pair $(r, \phi_0\sigma)$ *consisting of*

- *a rule* $r : \phi_0 \rightsquigarrow \phi_1 | \cdots | \phi_n$ *and*
- *a formula* $\phi_0\sigma$ *where* σ *is a substitution* $\sigma : \mathcal{V}(\phi_0) \to \mathcal{V}$

an index k, $1 \leq k \leq n$.

To each selection function sel *a fixpoint operator* $T_{\mathsf{sel},\mathsf{Pgm}}$ *on fact bases is associated. If* Pgm *or* sel *is clear from the context the corresponding index may be dropped. The operator is defined as follows:*[6] *If* $\mathcal{A}$ *is inconsistent,* $T_{\mathsf{sel}}(\mathcal{A}) = \mathcal{A}$, *otherwise*

$$T_{\mathsf{sel}} : \mathcal{A} \longmapsto \mathcal{A} \cup \left\{ \phi \;\middle|\; \begin{array}{l} \text{there is a renaming } \phi_0 \rightsquigarrow \phi_1 | \cdots | \phi_n \text{ with} \\ \text{fresh variables of a rule } r \in \mathsf{Pgm}, \\ \sigma : \mathcal{V}(\phi_0) \to \mathcal{V}(\mathcal{A}),\ \mathcal{A} \text{ entails } \phi_0\sigma, \\ \mathsf{sel}(r, \phi_0\sigma) = k, \phi = \left\{ \begin{array}{ll} \phi_k\sigma, & \text{if } k \neq 0 \\ \bot, & \text{otherwise} \end{array} \right. \end{array} \right\}$$

The iterated applications of the operator to a fact base $\mathcal{A}_0$ *are abbreviated as follows:*

$$\begin{array}{lcl} T_{\mathsf{sel},\mathsf{Pgm},\mathcal{A}_0}\uparrow^0 & := & \mathcal{A}_0, \\ T_{\mathsf{sel},\mathsf{Pgm},\mathcal{A}_0}\uparrow^{i+1} & := & T_{\mathsf{sel},\mathsf{Pgm}}(T_{\mathsf{sel},\mathsf{Pgm},\mathcal{A}_0}\uparrow^i), \text{ for } i > 0, \text{ and} \\ T_{\mathsf{sel},\mathsf{Pgm},\mathcal{A}_0}\uparrow^\omega & := & \bigcup_{i>0} T_{\mathsf{sel},\mathsf{Pgm},\mathcal{A}_0}\uparrow^i . \end{array}$$

The indices may be dropped, if they are clear from the context. $T\uparrow^\omega$ *is the* answer to $\mathcal{A}_0$ computed by Pgm with sel. □

[6] The phrase 'a renaming r_1 with fresh variables of a rule r' means that the variables of the rule r—as usual—have been substituted by pairwise different new variables which do not occur in $\mathcal{A}$. More precisely, there exists a bijection $\pi : \mathcal{V}(r) \to \mathcal{V}(r_1)$ such that $r\pi = r_1$ and $\mathcal{V}(r_1) \cap \mathcal{V}(\mathcal{A}) = \emptyset$.

Neglecting the incompleteness of the operational semantics which is due to the depth-first search strategy, it should be clear that the operational semantics of Section 5.1.1 can be understood as an implementation of the fixpoint semantics described here. For a given fact base $\mathcal{A}_0$ and a program Pgm, the incompleteness caused by the depth-first search strategy may only be problematic if there exists a selection function sel such that $T_{\mathsf{sel},\mathsf{Pgm},\mathcal{A}_0}\uparrow^\omega$ is infinite and consistent.

5.4 Soundness and Completeness Results

This section states and proves soundness and completeness of the fixpoint semantics with respect to the epistemic semantics of the program.

Proposition 5.4.1 (soundness) *Let a fact base $\mathcal{A}_0$, a program* Pgm, *and a selection function* sel *be given. Then each epistemic model of $T_{\mathsf{sel},\mathcal{A}_0,\mathsf{Pgm}}\uparrow^\omega$ satisfies $A_0 \cup \mathsf{Pgm}$.*

Proof. If $T\uparrow^\omega$ is inconsistent, the proposition holds trivially. If not, let $\mathcal{M}$ be an epistemic model of the consistent fact base $T\uparrow^\omega$. Since T is extending,[7] $\mathcal{A}_0 \subseteq T\uparrow^\omega$ and, thus, $\mathcal{M}$ satisfies $\mathcal{A}_0$.

It remains to show that $\mathcal{M}$ satisfies Pgm. Assume not. Then there is a rule $r : \forall x_0(B\phi_0) \Rightarrow \bigvee_{i=1\cdots n} \exists x_i(B\phi_i)$ in Pgm such that $\mathcal{M}$ does not satisfy r. I.e., there exists β such that $\mathcal{I},\beta \models \phi_0$, for all $\mathcal{I} \in \mathcal{M}$ and for all $i \in \{1,\cdots,n\}$ there exists some $\mathcal{I}' \in \mathcal{M}$ such that $\mathcal{M},\mathcal{I}',\beta \not\models \exists x_i(B\phi_i)$.

By Proposition 5.2.7, there exists a pre assignment α such that $\mathcal{M},\mathcal{I},\alpha \models T\uparrow^\omega$ and, with $\sigma := \beta\alpha^{-1}$, the set of condition formulas $T\uparrow^\omega$ entails $\phi_0\sigma$.

Since CF is compact, there exists a finite subset $\mathcal{A} \subseteq T\uparrow^\omega$ such that $\mathcal{A}$ entails $\phi_0\sigma$. Consequently, there exists a $j > 0$ such that $\mathcal{A} \subseteq T\uparrow^j$ and, thus, $T\uparrow^j$ entails $\phi_0\sigma$. Using the definition of T it follows that $\phi_k\sigma \in T\uparrow^{j+1} \subseteq T\uparrow^\omega$, for some k, and, thus, $\mathcal{M},\mathcal{I},\alpha \models \exists x_k(B\phi_k\sigma)$, for all $\mathcal{I} \in \mathcal{M}$. Finally, since $\sigma = \beta\alpha^{-1}$ and $\mathcal{I}' \in \mathcal{M}$, $\mathcal{M},\mathcal{I}',\beta \models \exists x_k(B\phi_k)$. Contradiction. □

Proposition 5.4.2 (completeness) *Let a fact base $\mathcal{A}_0$, a program* Pgm, *and an epistemic model $\mathcal{M} \subseteq \Gamma(D)$ of $A_0 \cup \mathsf{Pgm}$ be given. Then there exists a selection function* sel, *such that $\mathcal{M}$ is an epistemic model of $T_{\mathsf{sel},\mathcal{A}_0,\mathsf{Pgm}}\uparrow^\omega$.*

Proof. The proof of Proposition 3.7.5 is based on the following lemma which is a weak form of the proposition.

Lemma 5.4.3 (weak completeness) *Let a fact base $\mathcal{A}_0$, a program* Pgm, *and an epistemic model $\mathcal{M} \subseteq \Gamma(D)$ of $A_0 \cup \mathsf{Pgm}$ be given. Then there exists a selection function* sel, *such that $\mathcal{M}$ satisfies $T_{\mathsf{sel},\mathcal{A}_0,\mathsf{Pgm}}\uparrow^\omega$.*

[7] I.e., $\mathcal{A} \subseteq T(\mathcal{A})$ for all fact bases $\mathcal{A}$

Proof. In the proof a selection function sel is defined step by step as it is needed in the iterated application of the operator T_{sel}.

According to the assumption there exists a pre assignment $\alpha_0 : \mathcal{V}(\mathcal{A}_0) \to D$ such that $\mathcal{I}, \alpha_0^{\mathcal{I}} \models \mathcal{A}_0$, for each $\mathcal{I} \in \mathcal{M}$.

Induction hypothesis: $\mathcal{I}, \alpha_i^{\mathcal{I}} \models T\uparrow^i$, for each $\mathcal{I} \in \mathcal{M}$. Consider the $(i+1)$th iteration of the T operator and let σ, a rule $r \in \mathsf{Pgm}$, and its renaming with fresh variables $\phi_0 \rightsquigarrow \phi_1 | \cdots | \phi_n$ be given as in the definition of T, that is, $T\uparrow^i$ entails $\phi\sigma$. It is also assumed that sel has not been defined at $(r, \phi_0\sigma)$ so far.

The induction hypothesis implies $\mathcal{I}, \alpha_i^{\mathcal{I}} \models \phi\sigma$, for all $\mathcal{I} \in \mathcal{M}$. Then $\mathcal{I}, \sigma\alpha_i^{\mathcal{I}} \models \phi$, for all $\mathcal{I} \in \mathcal{M}$. Since $\mathcal{M} \models r$, this implies, $\mathcal{M}, \mathcal{I}, \sigma\alpha_i \models \bigvee_{i=1\cdots n} \exists x_i(B\phi_i)$. If $n = 0$, $\mathcal{M}$ would be empty and the lemma would hold trivially.

Otherwise, the proof proceeds as follows: Using the definition of disjunction it can be derived that there is a k, $1 \leq k \leq n$, such that $\mathcal{M}, \mathcal{I}, \sigma\alpha_i \models \exists x_k(B\phi_k)$ and, as an immediate consequence, $\mathcal{M}, \mathcal{I}, \alpha_i \models \exists x_k(B\phi_k\sigma)$. By the definition of $\models$ it follows that there is a pre assignment $\beta_{r,\sigma} : \mathcal{V}(\phi_k\sigma) \setminus \mathcal{V}(\phi_0\sigma) \to D$ such that $\mathcal{I}, (\alpha_i\beta_{r,\sigma})^{\mathcal{I}} \models \phi_k\sigma$, for all $\mathcal{I} \in \mathcal{M}$. Note, that $\beta_{r,\sigma}$ is defined on fresh variables. The value of $\mathsf{sel}(r, \phi_0\sigma)$ is set to k.

This construction can be made for all r, σ that trigger a rule in this iteration. The pre assignments $\beta_{r,\sigma}$ can be chosen such that they have pairwise disjoint domains. Hence, by identifying the pre assignments with their graph, α_{i+1} can be defined as $\alpha_i \cup \bigcup_{r,\sigma} \beta_{r,\sigma}$. Finally, $\mathcal{I}, \alpha_{i+1}^{\mathcal{I}} \models T\uparrow^{i+1}$, for each $\mathcal{I} \in \mathcal{M}$, because $\mathcal{I}, \alpha_i^{\mathcal{I}} \models T\uparrow^i$ and, $\mathcal{I}, \beta_{r,\sigma}^{\mathcal{I}} \models \phi\sigma$, for $\phi_{\mathsf{sel}(r,\phi_0\sigma)}\sigma \in T\uparrow^{i+1} \setminus T\uparrow^i$. □

With this lemma the proposition can be proved as follows: Let $\mathcal{A}_0$, Pgm, and an epistemic model $\mathcal{M}$ of $\mathcal{A}_0 \cup \mathsf{Pgm}$ be given. Then Lemma 5.4.3 implies that there exists a selection function sel such that $\mathcal{M}$ satisfies $T\uparrow^\omega$. By Observation 5.2.8 there exists an epistemic model $\mathcal{M}' \subseteq \Gamma(D)$ of $T\uparrow^\omega$ with $\mathcal{M} \subseteq \mathcal{M}'$. Proposition 5.4.1 implies that $\mathcal{M}'$ satisfies $\mathcal{A}_0 \cup \mathsf{Pgm}$. The maximality of $\mathcal{M}$ as an epistemic model of $\mathcal{A}_0 \cup \mathsf{Pgm}$ implies $\mathcal{M} = \mathcal{M}'$. Hence, $\mathcal{M}$ is an epistemic model of $T\uparrow^\omega$, too. □

The soundness result (Proposition 5.4.1) cannot be strengthened such that each epistemic model of $T\uparrow^\omega$ is also an epistemic *model* of $\mathcal{A}_0 \cup \mathsf{Pgm}$ as the following example shows.

Example 5.4.4 Let $p \rightsquigarrow a|b$ and $q \rightsquigarrow b|a$ be a program and $\mathcal{A}_0 = \{p, q\}$. Then $T\uparrow^\omega = \{p, q, a, b\}$, for some selection function. But, the epistemic models of $T\uparrow^\omega$ do not correspond to maximal epistemic interpretations satisfying $\mathcal{A}_0$, because already the smaller fact bases $\{p, q, a\}$ and $\{p, q, b\}$ are sound answers that have larger epistemic models. □

In the general case, it is computationally very expensive (in an informal sense) to check whether a selection function leads to a computed answer that is redundant in a way analogue to the example. For each formula that may be added to the fact base it would be necessary to check, whether one of the alternatives of this formula is already entailed by the fact base and, even worse, it would have to be checked whether the newly introduced formula makes previous selections obsolete because the new fact base would entail another alternative.

Example 5.4.5 Consider, $A_0 := \{p, q\}$ with the rules $p \rightsquigarrow a|b$ and $q \rightsquigarrow b$. Now assume that the first rule is considered first and that a has been selected. In a later stage of the computation when the second rule is considered the first choice gets obsolete since at this moment it gets obvious that b is necessarily contained in the computed answer. □

5.5 Relation to Horn Logic and CLP Formalisms

In this section, it will first be shown how Horn logic programs can be translated to the epistemic rule formalism. The section concludes by discussing the relation to CLP languages as described in [Höhfeld and Smolka, 1988; Jaffar and Lassez, 1986]. Note that the primary intention for the translations developed in this section, is to provide the reader with a better intuition of the capabilities of the rule formalism and to demonstrate how goal-directed and data-driven inferences can be represented, in principle.

5.5.1 Some Relations to Horn Logic

If either bottom-up, or top-down reasoning is associated with the rules this operational aspect can be represented in the epistemic formalism. First note that a Herbrand term $f(t_1, t_2)$ can be represented by $f(x_1, x_2)$, $x_1 = t_1$, $x_2 = t_2$. So, in order to represent Horn logic programs, it suffices to consider a condition formalism with the following kind of formulas:

1. Atoms of the form $p(x_1, \cdots, x_n)$ where the x_i are pairwise different variables and p is a predicate name.

2. Equations $x_1 = x_2$ and $x_0 = f(x_1, \cdots, x_n)$ where the x_i are pairwise different variables and f is an n-ary function symbol.

These formulas are interpreted as usual in first-order logic with equality. Note that a fact base $\mathcal{A}$ entails $p(x_1, \cdots, x_n)$ if $p(x_1, \cdots, x_n) \in \mathcal{A}$.

Top-down Rules

Let be given a Horn logic program and an n-ary predicate p where

$$p(x) \leftarrow G_1 \vee \cdots \vee G_n$$

is the *homogeneous form*[8] of the rules defining the predicate p. These rules are then translated to a single rule $p(x) \rightsquigarrow G_1' | \cdots | G_n'$ where each G_i' is obtained from G_i by translating terms into sets of equations.

Let a goal G and substitution σ be given. Then σ is a correct answer substitution for G [Lloyd, 1987] iff there exists a selection function sel such that $T_{\mathsf{sel}}\uparrow^\omega$ is finite and σ restricted to the variables in G can be extended to a unifier of $T_{\mathsf{sel}}\uparrow^\omega$.

Example 5.5.1 The append program

$$\begin{aligned}
\mathsf{append}(X, Y, Z) \leftarrow\ & X = \mathsf{nil}, Y = Z. \\
\mathsf{append}(X, Y, Z) \leftarrow\ & X = \mathsf{cons}(Car, Cdr), \\
& Z = \mathsf{cons}(Car, Z'), \\
& \mathsf{append}(Cdr, Y, Z').
\end{aligned}$$

is translated into

$$\begin{aligned}
\mathsf{append}(X, Y, Z) \rightsquigarrow\ & X = \mathsf{nil}, Y = Z \\
| \ & X = \mathsf{cons}(Car, Cdr), \\
& Z = \mathsf{cons}(Car, Z'), \\
& \mathsf{append}(Cdr, Y, Z').
\end{aligned}$$

For the fact base (i.e., query) $\mathsf{append}(X, Y, Z)$, $X = \mathsf{cons}(A, nil)$, $Y = \mathsf{cons}(C, D)$ the applications of the T operator add the following sets (for an appropriate selection function):

1. Selecting the second (alternative of the) head, the formulas

   $$X = \mathsf{cons}(Car, Cdr), \quad Z = \mathsf{cons}(Car, Z'), \quad \mathsf{append}(Cdr, Y, Z')$$

   are added. Together, with the initial query this would imply $Car = A$, $Cdr = \mathsf{nil}$, $Z = \mathsf{cons}(A, Z')$.

2. Selecting the second head again would lead to an inconsistency whereas selecting the first head adds $Cdr = \mathsf{nil}$, $Y = Z'$.

At this point T gets stable and a most general unifier for the equations in the fact base would assign $\mathsf{cons}(A, \mathsf{cons}(C, D))$ to Z. □

If all predicates are translated in this manner, all queries are answered just as if SLD resolution would have been used—with one difference: *All* atoms and conditions on variables occurring in the SLD-derivations are kept and the intermediate goals are considered as sets.

[8] For example, the homogeneous form of $p(a) \leftarrow G$, $p(b) \leftarrow G'$ is $p(X) \leftarrow ((X = a \wedge G) \vee (X = b \wedge G'))$ where X is a fresh variable, not occurring in G or G'.

Bottom-up Rules

In this case the rules have to be range restricted (i.e., each variable in the head is bound in the body). [9] Each rule $H \leftarrow G$ is just translated to $G \rightsquigarrow H$. Obviously, then the operational semantics of the epistemic rule formalism directly implements naive bottom-up computation.

See [Hinkelmann, 1993] for a more evolved discussion of *consequence finding strategies* [Slagle *et al.*, 1969; Minicozzi and Reiter, 1972; Inoue, 1991] with **mixed top-down/bottom-up** computation in the context of logical programming.

5.5.2 Relation to CLP

In the CLP schema proposed in [Höhfeld and Smolka, 1988] and refined in [Smolka, 1989] a CLP clause is of the form

$$p_0(x_0) \leftarrow p_1(x_1), \cdots, p_n(x_n) \& \phi \tag{5.2}$$

where p_i are "relational predicates", x_i are tuples of pairwise different variables, and ϕ is a constraint (i.e., a formula of a first-order logic satisfying certain requirements). Atoms over relational predicates and constraints are disjoint classes of formulas. A query to a set of such clauses is a conjunction of atoms and constraints. The operational semantics (see, for example, [Smolka, 1989]) then tries to enumerate constraints such that if a variable assignment of a model of the program satisfies an answer constraint, the query is satisfied, too. If for a query, the operational semantics detects in finite time that there is only the trivial answer $\bot$ that entails the query, the query belongs to the *finite failure* set of the program.

Relational versus Constraint Predicates

In this formalization of CLP the knowledge engineer is forced to decide which portion of knowledge to represented by which kind of predicates. More recent CLP formalisms (see for example [Würtz *et al.*, 1993]) tend to blur the distinction between these two kinds of predicates—the formalisms only deals with constraints. The semantics may then be defined in terms of a closure of certain constraint propagation and simplification operations. These formalisms can also be given a semantics based on classical logic. In general, the operational semantics is sound, but not complete with respect to these semantics.

The rule formalism introduced here, is also uniform in the sense that there is only one class of formulas: condition formulas. But here, this is also reflected by the model-theoretic semantics.

[9] To relax this requirement, explicit universal quantification of the variables in a fact base would be necessary. See also the note on 'filtering' after Definition 5.2.6.

Translation

The CLP clauses defining a predicate of a CLP program P, can be translated analogue to the case of top-down rules in Section 5.5.1 into an epistemic program P'. I.e., the homogeneous form of a predictate definition[10]

$$p(x) \leftarrow (G_1 \& \phi_1) \vee \cdots \vee (G_k \& \phi_k) \tag{5.3}$$

is translated into

$$p(x) \rightsquigarrow (G_1 \& \phi_1) | \cdots | (G_k \& \phi_k) \tag{5.4}$$

The entailment test is defined as follows: A fact base $\mathcal{A}$ entails $p(x)$:iff $p(x) \in \mathcal{A}$. The consistency test is just the consistency test of the CLP formalism.[11]

Observation 5.5.2 *Let P be a CLP program and G a CLP goal. If P' and G' are the respective translations then G belongs to the finite failure set of P iff $G' \cup P'$ does not have an epistemic model.* □

Note that a condition formalism is always compact. Hence, the above observation can only be made for CLP formalisms with a compact constraint formalisms. Equations of Herbrand terms which are only interpreted over Herbrand terms (as in Prolog) do not induce a compact constraint formalisms: The set of equations

$$\{X_0 = f(X_1), X_1 = f(X_2), \cdots\} \tag{5.5}$$

is inconsistent although each finite subset is consistent.

5.6 Integrating Terminological Reasoning, Concrete Domains and the Rule Formalism

In this section it is shown how the generic rule scheme can be combined with the generic terminological formalism ALCFP($\mathcal{D}$). After some preliminary discussion how ALCFP($\mathcal{D}$) can be considered as a condition formalism (cf. Section 5.6.1) the rule scheme is applied to ALCFP($\mathcal{R}$). On the basis of this three layered formalism (rules on top of a terminological formalism on top of concrete domains) the representation and reasoning demands left open in Section 4.2 are reconsidered in Section 5.6.2. In particular, it is demonstrated how aggregation, derived attributes, and sequences can be dealt with.

[10]The G_i are conjunctions of relational predicates with pairwise different variables, and the ϕ_i are constraint formulas.

[11]More precisely, the relational predicates have to be dropped before the consistency test can be applied to the constraints.

5.6.1 Application of the Scheme to ALCFP($\mathcal{D}$)

The formulas of the condition formalism are finite A-boxes. More precisely, a fact base is a set of generalized membership assertions where the occurring objects are considered as variables. An alternative in a head is also a collection of generalized membership assertions. Premises are filtering[12] A-boxes (i.e., also collections of generalized membership assertions) which are rooted by some objects $y_1, \cdots, y_n$. These objects are the free variables in the premise. [13]

According to Chapter 3 the domain of a terminological interpretation is divided into disjoint sets: an abstract domain $\text{DOM}_{\mathcal{I}}$ and a concrete domain $\text{DOM}_{\mathcal{D}}$. To avoid a name clash with the domain of an interpretation of a condition formalism the abstract domain is denoted by $\text{DOM}_{\mathcal{I},a}$ from here on. In the original definition the set $\text{DOM}_{\mathcal{D}}$ remains the same for a given concrete domain $\mathcal{D}$ and does not depend on $\mathcal{I}$. This conflicts with the definition of an admissible interpretation domain of a condition formalism.

Hence, in the context of the rule formalism an interpretation domain is of the form $\text{DOM}_{\mathcal{I},a} \cup \text{DOM}_{\mathcal{I},\mathcal{D}}$ and is denoted by $\text{DOM}_{\mathcal{I}}$. Note that the concrete domain is denoted by $\text{DOM}_{\mathcal{I},\mathcal{D}}$ to express that it depends on $\mathcal{I}$. It is also required that the 'abstract domain' $\text{DOM}_{\mathcal{I},a}$ and the 'concrete domain' $\text{DOM}_{\mathcal{I},\mathcal{D}}$ are disjoint, which is not problematic with respect to the definition of a condition formalism. The relation of $\text{DOM}_{\mathcal{D}}$ to $\text{DOM}_{\mathcal{I},\mathcal{D}}$ is established by requiring that for each interpretation there is a bijection $\pi_{\mathcal{I}} : \text{DOM}_{\mathcal{D}} \rightarrow \text{DOM}_{\mathcal{I},\mathcal{D}}$ and the interpretation of a concrete predicate $p(x_1, \cdots, x_n)$ is defined by $p^{\mathcal{I}} := \{(e_1\pi, \cdots, e_n\pi) |\ (e_1, \cdots, e_n) \in p^{\mathcal{D}}\}$. With these modifications, the domain of $\text{DOM}_{\mathcal{I}}$ can still be a partition with infinite equivalence classes, as it is required by the definition of an epistemic interpretation.

The definition of a rule requires that the premise should be filtering. An example of a premise that is not filtering is $x : \top$, if $\text{DOM}_{\mathcal{D}} = \emptyset$. The more complex premise $x : (\top \sqcup \forall\epsilon.\mathcal{D})$ is not filtering also if $\text{DOM}_{\mathcal{D}} \neq \emptyset$. A premise ϕ is not filtering with respect to a fact base $\mathcal{A}$ and a variable $x \in \mathcal{V}$ if there exists a substitution σ such that $\mathcal{A}$ entails $\phi\sigma$ and $x\sigma$ is a fresh variable. But a premise ϕ is filtering with respect to a variable and all fact bases, for example,

- if x is a member of a concept C that is not equivalent to $x : (\top \sqcup \forall\epsilon.\mathcal{D})$,
- if x occurs in a concrete or abstract predicate assertion,
- if x occurs in a role/attribute filler assertion.

Theorem 3.5.7 implies that the A-box consistency test may check effectively whether a fact base is consistent and that the A-box subsumption test may check effectively whether a fact base $\mathcal{A}$ entails an instantiated premise $\phi\sigma$. Hence, the rule

[12] Discussed below.

[13] To avoid writing quantifiers, objects in the premise that are not intended to participate in rooting the A-box may be marked by a leading "_".

formalism instantiated with ALCFP($\mathcal{D}$) where $\mathcal{D}$ is an admissible concrete domain can be implemented, for example, as sketched in Section 5.1.1.

5.6.2 Limitations Revisited

To reconsider the limitations of Section 4.2 the rule scheme is applied to ALCFP($\mathcal{R}$) where $\mathcal{R}$ is the concrete domain of real numbers. With the additional expressiveness the representation and reasoning requirements left open can be satisfied. It should be noted that the rule formalism is a semidecidable formalism. Hence the knowledge engineer is responsible of the termination of the inference algorithms for the intended queries. This task is facilitated by the easy to understand, intuitive operational semantics of the rule formalism and the fact that the terminological reasoning steps initiated by the rule level always terminate and have a declarative semantics.

Aggregation

The problem is to formulate when objects are aggregated and to introduce an object that corresponds to an aggregate when the condition of an aggregation are satisfied. In the example of Section 4.2.1 two truncated cones have to be aggregated if they are neighbored. This can be expressed by the following rule:

$$\begin{array}{l} X : \mathsf{truncone} \wedge \\ Y : \mathsf{truncone} \wedge \\ (X,Y) : \mathsf{neighbored} \rightsquigarrow \begin{array}{l} (N,X) : \mathsf{left} \wedge \\ (N,Y) : \mathsf{right} \end{array} \end{array} \tag{5.6}$$

If this rule is applied to the A-box comprising (4.1) and (4.2), a new object N would be introduced that can be classified as a rshoulder.

Sometimes one would like to represent that an aggregate is completely determined by its components. In the example this can be done by the following rule, which would be problematic if the terminological formalism have had a unique-name assumption.

$$\begin{array}{l} (B_1,L_1) : \mathsf{left} \wedge (B_1,R_1) : \mathsf{right} \wedge \\ (B_2,L_2) : \mathsf{left} \wedge (B_2,R_2) : \mathsf{right} \wedge \\ L_1 = L_2 \wedge R_1 = R_2 \rightsquigarrow B_1 = B_2 \end{array} \tag{5.7}$$

Derived Attributes

Similar to aggregation, it can be formulated in the premise of a rule when a new attribute or role filler has to be introduced. In the example of Section 4.2.2 the problem is to introduce height and width of a biconic once it is known that it is a shoulder. All that has to be done is to add the following simple rule.

$$B : \mathsf{shoulder} \rightsquigarrow B : \mathsf{rshoulder\text{-}with\text{-}hw} \tag{5.8}$$

Note that it is not necessary to know the values of the radii and the centers of the involved truncated cone.

Sequences

It has already been shown in Section 4.2.3 how it can be represented that consecutive truncated cones in the sequence are connected. Since rules may be recursive, the varying length aspect of a sequence can easily be represented, here. The following rule is a definition of the concept 'sequence of truncated cones' from a goal directed point of view à la Prolog.

$$\begin{aligned} S : \textsf{seq-of-tc} \quad \rightsquigarrow \quad & (S,H) : \textsf{head} \wedge \\ & H : \textsf{truncone} \wedge \\ & S : \textsf{last} \\ | \quad & S : \textsf{connected} \wedge \\ & (S,H) : \textsf{head} \wedge \\ & (S,S') : \textsf{tail} \wedge \\ & H : \textsf{truncone} \wedge \\ & S' : \textsf{seq-of-tc} \end{aligned} \tag{5.9}$$

According to the operational semantics a given fact B : seq-of-tc is recursively expanded. Using terminological constructs a much more compact definition can be obtained:

$$\begin{aligned} S : \textsf{seq-of-tc} \quad \rightsquigarrow \quad & S : \textsf{connected} \wedge \\ & S : \exists \textsf{head.truncone} \wedge \\ & S : \forall \textsf{tail.seq-of-tc} \end{aligned} \tag{5.10}$$

The definition of an 'ascending sequences of truncated cones' aseq-of-tc is analogue:

$$\begin{aligned} S : \textsf{aseq-of-tc} \quad \rightsquigarrow \quad & S : \textsf{connected} \wedge \\ & S : \exists \textsf{head.asc-tc} \wedge \\ & S : \forall \textsf{tail.aseq-of-tc} \end{aligned} \tag{5.11}$$

If one is interested in finding sequences in a given fact base from elementary data, this can be represented by the following two (production rule-like) rules.

$$\begin{aligned} H : \textsf{truncone} \rightsquigarrow \quad & (S,H) : \textsf{head} \wedge \\ & S : \textsf{seq-of-tc} \wedge \\ & S : \textsf{last} \end{aligned} \tag{5.12}$$

$$\begin{aligned} & H : \textsf{truncone} \wedge \\ & S' : \textsf{seq-of-tc} \wedge \\ & (S',H') : \textsf{head} \wedge \\ & (H,H') : \textsf{neighbored} \quad \rightsquigarrow \quad S : \textsf{seq-of-tc} \wedge \\ & \qquad\qquad\qquad\qquad\quad (S,H) : \textsf{head} \wedge \\ & \qquad\qquad\qquad\qquad\quad (S,S') : \textsf{tail} \end{aligned}$$

Note that these rules contain more knowledge then the 'goal-directed versions' (5.9 ,5.10). They explicitly represent (in the premise of the second rule) *when* a truncated

cone and an already existing sequence of truncated cones have to be aggregated to a longer sequence and they describe in the head *what* has to be added to the fact base for aggregation.

It should be noted that the reversed rules (5.13) of (5.12) can also be used to traverse an existing sequence of truncated cones similar to (5.9). The difference is that the reversed rules (5.13) terminate in more cases.

$$\begin{array}{ll} (S,H) : \textsf{head} \wedge & \\ S : \textsf{seq-of-tc} \wedge & \\ S : \textsf{last} & \rightsquigarrow H : \textsf{truncone} \end{array} \tag{5.13}$$

$$\begin{array}{ll} S : \textsf{seq-of-tc} \wedge & \\ (S,H) : \textsf{head} \wedge & \\ (S,S') : \textsf{tail} & \rightsquigarrow H : \textsf{truncone} \wedge \\ & \quad S' : \textsf{seq-of-tc} \wedge \\ & \quad (S',H') : \textsf{head} \wedge \\ & \quad (H,H') : \textsf{neighbored} \end{array}$$

However, the specialization of the goal-directed version of the definition of 'a sequence of truncated cones' to 'an ascending sequence of truncated cones' is very simple. If the rules (5.12), which encode the aggregation knowledge, are in the program, it is sufficient to add the following rule:

$$\begin{array}{ll} S : \forall\textsf{tail.aseq-of-tc} \wedge & \\ S : \forall\textsf{head.asc-tc} & \rightsquigarrow S : \textsf{aseq-of-tc} \end{array} \tag{5.14}$$

Chapter 6

Summary

In his invited talk at the 8^{th} National Conference on AI in 1990 [Brachman, 1990] Ron Brachman argued that the development of "unified reasoners" is one potential highlight of the "future of knowledge representation" (p. 1089). Similarly, "incomplete reasoners" and "expressiveness vs. tractability" are mentioned in a list of important open research problems (p. 1090). This thesis has settled research on "unified reasoners" in a subfield of symbolic, logic-oriented knowledge representation to a certain extent. Terminological knowledge representation and reasoning can now be utilized for more realistic applications as an integral component of a hybrid knowledge representation formalism.

The structure of the thesis can be understood on the basis of the following simple observation: the expressiveness of a decidable knowledge representation formalism is limited.

Hence it has first been explored how the expressiveness of terminological formalisms could be maximized with respect to relevant representation demands of applications under the global requirement of decidable inferences. After a certain optimum has been found (for instance, the terminological formalism $\mathsf{ALCFP}(\mathcal{D})$), the idea was to delegate the remaining representation demands that could not be satisfied so far to an embedding formalism that is computationally complete.

For this purpose a scheme has been developed that constructs for a given decidable logic formalism CF satisfying certain requirements (such as $\mathsf{ALCFP}(\mathcal{D})$) a computationally complete rule formalism that contains CF as an integral component. In this generic rule formalism data-driven (bottom-up) and goal-directed (top-down) rule inferences have been integrated, because

- rule-based knowledge representation is a centerpiece of symbolic, logic-oriented knowledge representation,
- both data-driven and goal-directed inferences occur in realistic applications, and
- rules are suitable to represent varying-size aspects, which were problematic for terminological systems.

In the generic rule formalism the reasoning direction of each portion (so to say) of knowledge is explicit. This knowledge about reasoning direction constrains the search space and enables more efficient reasoning than in approaches where the embedding system is a general purpose theorem prover as for example in [Bürckert, 1991] or [Brachman *et al.*, 1983].

The price for the enhanced efficiency is the incompleteness of the operational semantics with respect to a reading of the rules in classical first-order logic. To compensate this deficit a model-theoretic semantics based on a minimal belief logic has been developed that characterizes this "incomplete reasoner". Existing formalizations of minimal belief logics for the first-order case had surprising deficits, which had to be resolved to obtain soundness and completeness results.

It should be noted that the mechanical engineering application considered in the ARC-TEC Project [Bernardi *et al.*, 1991] at the DFKI has been a rich, valuable source of representation and reasoning problems. The need for a simultaneous treatment of these problems in one application has led to this more integrated view of knowledge representation.

The following list provides pointers to the technical results of the thesis.

- The subsumption problem is undecidable for the conventional concept language ALCF extended by certain concept-forming operators for *role interaction* that are based on $=$ and $\neq$ and generalize attribute (dis)agreements in a manner different than role-value maps (Theorem 3.3.3).

- Generic decision procedures for the reasoning services of a concept language that is parametrized by a concrete domain and supports role interaction by attribute (dis)agreements, abstract predicates and concrete predicates (Theorems 3.4.5, 3.5.7, and algorithm in Figure 3.5).

- A new reasoning service, called *A-box subsumption* comparing A-boxes with respect to generality (Definition 3.5.3). A-box subsumption is undecidable in general (Proposition 3.5.4). It is decidable for rooted A-boxes (Definition 3.5.5). A-box subsumption is needed in the context of the generic rule formalism.

- A rule schema (Section 5.3) that is parametrized by a condition logic (i.e., a first-order logic satisfying certain requirements) and that integrates data-driven and goal-directed reasoning on the basis of the inference algorithms of the condition formalism.

- A "really" (Section 5.1.2) minimal belief logic (Section 5.2) for the first-order case with modal operator B.

- Soundness and completeness (Section 5.4) of the operational semantics of the rule schema with respect to a model-theoretic semantics based on the "really" minimal belief logic.

- A declarative integration of terminological, constraint-based, data-driven and goal-directed reasoning (Section 5.6) obtained by an application of the rule scheme to the assertional formalism of a terminological formalism parametrized by a concrete domain.

Bibliography

[Abecker and Hanschke, 1993] A. Abecker and P. Hanschke. TaxLog: A flexible architecture for logic programming with structured types and constraints. In *Notes of the Workshop on Constraint Processing held in conjunction with CSAM'93, Petersburg*, 1993.

[Aït-Kaci and Nasr, 1986] H. Aït-Kaci and R. Nasr. LOGIN: A logic programming language with built-in inheritance. *The Journal of Logic Programming*, 3, 1986.

[Aït-Kaci and Podelski, 1991] H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. In J. Maluszyński and M. Wirsing, editors, *Proceedings of the 3rd Int. Symposium on Programming Language Implementation and Logic Programming, PLILP91, Passau, Germany*, pages 255–274. Springer Verlag, August 1991.

[Baader and Hanschke, 1991a] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In Mylopoulos and Reiter [1991].

[Baader and Hanschke, 1991b] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. Research report RR-91-10, DFKI, 1991. Extended version with proofs and algorithms.

[Baader and Hanschke, 1992] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In Ohlbach [1992].

[Baader *et al.*, 1991] F. Baader, H.-J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. Technical report, DFKI, 1991.

[Baader, 1990] F. Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, volume 2, pages 621–626, 1990.

[Baader, 1991] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In Mylopoulos and Reiter [1991].

[Bernardi *et al.*, 1991] A. Bernardi, H. Boley, K. Hinkelmann, P. Hanschke, C. Klauck, O. Kühn, R. Legleitner, M. Meyer, M. M. Richter, G. Schmidt, F. Schmalhofer, and W. Sommer. ARC-TEC: Acquisition, Representation and

Compilation of Technical Knowledge. In *Expert Systems and their Applications: Tools, Techniques and Methods*, 1991.

[Bernardi *et al.*, 1992a] A. Bernardi, C. Klauck, and R. Legleitner. FEAT-REP: Representing feature languages in CAD/CAM. In Ko and Tan [1992].

[Bernardi *et al.*, 1992b] A. Bernardi, C. Klauck, and R. Legleitner. PIM: Skeletal plan based CAPP. In Ko and Tan [1992].

[Boley *et al.*, 1993] H. Boley, P. Hanschke, K. Hinkelmann, and M. Meyer. COLAB: A hybrid knowledge representation and compilation laboratory. *Annals of Operations Research*, 1993. forthcoming.

[Boone, 1959] W. W. Boone. The word problem. *Ann. of Mat.*, 1959.

[Borgida *et al.*, 1989] A. Borgida, R. Brachman, D. McGuinness, and L. Resnick. CLASSIC: A structural data model for objects. In *International Conference on Managment on Data*. ACM SIGMOD, 1989.

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

[Brachman *et al.*, 1983] R. J. Brachman, R. E. Fikes, and H. J. Levesque. KRYPTON: Integrating terminology and assertion. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 31–35. AAAI, August 1983.

[Brachman *et al.*, 1991] R. Brachman, D. McGuinness, P. Pate-Schneider, and L. Resnick. Living with CLASSIC: When and how to use a KL-ONE-like language. In *Principles of Semantic Networks*. Morgan Kaufmann, 1991.

[Brachman, 1990] R. L. Brachman. The future of knowledge representation. In *Proceedings Eight National Conference on Artificial Intelligence*, volume two, pages 1082–1092. AAAI, The MIT Press / AAAI Press, 1990.

[Bürckert, 1991] H.-J. Bürckert. *A Resolution Principel for a Logic with Restricted Quantifiers*. Number 568 in LNAI. Springer, 1991.

[Clancey, 1985] W. J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[Collins, 1975] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *2nd Conference on Automata Theory & Formal Languages*, volume 33, 1975.

[Dershowitz and Manna, 1979] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the* ACM, 8(22):465–476, 1979.

[Donini *et al.*, 1991] F. Donini, D. Lenzerini, D. Nardi, and W. Nutt. Adding epistemic operators to concept languages. In Mylopoulos and Reiter [1991].

[Donini *et al.*, 1992] F. Donini, D. Lenzerini, A. Schaerf, and W. Nutt. Adding epistemic operators to concept languages. In Nebel et al. [1992].

[Edelmann and Owsnicki, 1986] J. Edelmann and B. Owsnicki. Data models in knowledge representation systems: A case study. In *GWAI-86 and 2. Österreichische Artificial-Intelligence Tagung*, pages 69–74. Springer, 1986.

[Firebaugh, 1988] M. W. Firebaugh. *Artificial Intelligence, A Knowledge-Based Approach*, chapter 13.3. PWS-KENT Publisher Company, Boston, 1988.

[Frühwirth and Hanschke, 1993] T. Frühwirth and P. Hanschke. Terminological reasoning with constraint handling rules. In *Position Papers for the First Workshop on Principles and Practice of Constraint Programming, Newport, Rhode Island*, April 1993. A preliminary version is available as DFKI Document DD-93-01.

[Hanschke and Hinkelmann, 1992] P. Hanschke and K. Hinkelmann. Combining terminological and rule-based reasoning for abstraction processes. In Ohlbach [1992].

[Hanschke and Meyer, 1992] P. Hanschke and M. Meyer. An alternative to θ-subsumption based on terminological reasoning. In C. Rouveirol, editor, *Workshop on Logical Approaches to Machine Learning, ECAI 92, Vienna*, August 1992.

[Hanschke and Würtz, 1993] P. Hanschke and J. Würtz. Satisfiability of the smallest binary programm. *Information Processing Letters*, 45, April 1993.

[Hanschke, 1992] P. Hanschke. Specifying role interaction in concept languages. In Nebel et al. [1992].

[Hinkelmann, 1993] K. Hinkelmann. Consequence finding and logic programming. Presented at the workshop "Neuere Entwicklungen der Deklarativen KI-Programmierung" held at 17. Fachtagung für Künstliche Intelligenz, 1993. A revised version will be available as DFKI report.

[Höhfeld and Smolka, 1988] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Lilog-Report 53, IBM Deutschland, October 1988.

[Hollunder *et al.*, 1990] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *9th European Conference on Artificial Intelligence (ECAI'90)*, 1990.

[Inoue, 1991] K. Inoue. Consequence-finding based on ordered linear resolution. In Mylopoulos and Reiter [1991].

[Jaakola, 1990] J. Jaakola. Modifying the simplex algorithm to a constraint solver. In P. Deransart and J. Małuszyński, editors, *Programming Languages Implementation and Logic Programming*, Linköping, Sweden, 1990. Springer-Verlag.

[Jaffar and Lassez, 1986] J. Jaffar and J.-L. Lassez. Constraint logic programming. Technical report, Monash University, Department of Computer Science, June 1986.

[Jaffar *et al.*, 1990] J. Jaffar, S. Michaylov, P. J. Stuckey, and Y. R. H. C. The CLP(R) language and system. Technical Report CMU-CS-9–181, Carnegie Mellon University, 1990.

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

[Ko and Tan, 1992] N. W. M. Ko and S. T. Tan, editors. *Proc. of the international conference on manufacturing automation.* University of Hong Kong, August 1992.

[Levesque and Brachman, 1987] H. J. Levesque and R. J. Brachman. Expressivity and tractability in knowledge representation and reasoning. *Computational Intelligence*, 1987.

[Levesque, 1984] J. Levesque, H. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 1984.

[Lifschitz, 1991] V. Lifschitz. Minimal belief and negation as failure. In Mylopoulos and Reiter [1991].

[Lloyd, 1987] J. W. Lloyd. *Foundations of Logic Programming.* Springer, Berlin, Heidelberg, New York, 1987.

[Loos and Weispfenning, 1990] R. Loos and V. Weispfenning. Applying linear quantifier elimination. Technical report, Wilhelm Schickard-Institut für Informatik, Universität Tübingen, Germany, 1990.

[MacGregor, 1988] R. MacGregor. A deductive pattern matcher. In *AAAI*, pages 403–408, 1988.

[Minicozzi and Reiter, 1972] E. Minicozzi and R. Reiter. A note on linear resolution strategies in consequence-finding. *Artificial Intellligence*, 3:175–180, 1972.

[Mylopoulos and Reiter, 1991] J. Mylopoulos and R. Reiter, editors. *12th International Joint Conference on Artificial Intelligence*, 1991.

[Nebel *et al.*, 1992] B. Nebel, C. Rich, and W. Swartout, editors. *Third International Conference on Principles of Knowledge Representation and Reasoning (KR '92).* Morgan Kaufmann, 1992.

[Nebel, 1990] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *LNAI*. Springer, 1990.

[Ohlbach, 1992] H.-J. Ohlbach, editor. *Proceedings of the 16th German AI-Conference (GWAI-92)*, volume 671 of *LNAI*. Springer, April 1992.

[Patel-Schneider, 1989] P. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39(2), 1989.

[Reiter, 1990] R. Reiter. On asking what a database knows. In *Computational Logic Symposium Proceedings*, November 1990.

[Schmalhofer *et al.*, 1991] F. Schmalhofer, O. Kühn, and G. Schmidt. Integrated knowledge acquisition from text, previously solved cases and expert memories and expert memories. *Applied Artificial Intelligence*, 5:311 – 337, 1991.

[Schmidt-Schauß and Smolka, 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Journal of Artificial Intelligence*, 48(1):1–26, 1991.

[Schmidt-Schauß, 1989] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *First International Conference On Principles of Knowledge Representation and Reasoning*, 1989.

[Slagle *et al.*, 1969] J. R. Slagle, C. L. Chang, and R. C. T. Lee. Completeness theorems for semantic resolution in consequence-finding. In *Proc. of the 1 st IJCAI*, pages 281–285, 1969.

[Smolka, 1989] G. Smolka. *Logic Programming over Polymorphically Order-Sorted Types*. PhD thesis, University of Kaiserslautern, Germany, 1989.

[Steinle, 1993] F. Steinle. HAMLET: Erweiterung eines Constraint-Systems um Negation und Disjunktion und dessen Anbindung an eine Konzeptbeschreibungssprache, March 1993. Projektarbeit, FB Informatik, Universität Kaiserslautern. In German.

[Tarski, 1951] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. U. of California Press. Berkley, 1951.

[van Hentenryck, 1989] P. van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, Ma., 1989.

[Weispfenning, 1988] V. Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5, 1988.

[Würtz *et al.*, 1993] J. Würtz, M. Henz, and G. Smolka. Oz - a programming language for multi-agent systems. In *IJCAI'93*, 1993. forthcoming.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

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

DFKI Publications

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

DFKI Research Reports

RR-92-45
Elisabeth André, Thomas Rist: The Design of Illustrated Documents as a Planning Task
21 pages

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster: WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

DFKI Technical Memos

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

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

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

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

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

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

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

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

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

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

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

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

DFKI Documents

D-92-23
Michael Herfert: Parsen und Generieren der Prolog-artigen Syntax von RELFUN
51 Seiten

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.): Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

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

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch: Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning

Philipp Hanschke

RR-93-46
Research Report