

A Scheme for Integrating Concrete Domains into Concept Languages

Franz Baader

Philipp Hanschke

German Research Center for AI (DFKI)

Postfach 2080

W-6750 Kaiserslautern, Germany

e-mail: baader@dfki.uni-kl.de, hanschke@dfki.uni-kl.de

phone: (+49 631)205-3457, (+49 631)205-3460

Abstract

A drawback which concept languages based on KL-ONE have is that all the terminological knowledge has to be defined on an abstract logical level. In many applications, one would like to be able to refer to concrete domains and predicates on these domains when defining concepts. Examples for such concrete domains are the integers, the real numbers, or also non-arithmetic domains, and predicates could be equality, inequality, or more complex predicates.

In the present paper we shall propose a scheme for integrating such concrete domains into concept languages rather than describing a particular extension by some specific concrete domain. We shall define a terminological and an assertional language, and consider the important inference problems such as subsumption, instantiation, and consistency. The formal semantics as well as the reasoning algorithms are given on the scheme level. In contrast to existing KL-ONE based systems, these algorithms will be not only sound but also complete. They generate subtasks which have to be solved by a special purpose reasoner of the concrete domain.

Contents

1	Introduction	3
2	Concrete Domains	5
3	The Concept Language	8
3.1	Syntax and Semantics	8
3.2	Terminological Reasoning	9
4	The Assertional Language	11
4.1	Syntax and Semantics	11
4.2	Assertional Reasoning	13
5	The Basic Reasoning Algorithm	13
6	Soundness and Completeness	16
7	Expressing Interval Relations: An Example	22
8	An Undecidability Result	24
9	Conclusion	27

1 Introduction

Concept languages based on KL-ONE [Brachman and Schmolze, 1985] are used to represent the taxonomical and conceptual 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 **child**. If the logical connective conjunction is present as language construct, one may describe the concept **Woman** as “humans who are female”, and represent it by the expression $\text{Human} \sqcap \text{Female}$. Many languages provide quantification over role fillers which allows for example to describe the concept **Mother** by the expression $\text{Woman} \sqcap \exists \text{child.Human}$.

KL-ONE was first developed for the purpose of natural language processing [Brachman *et al.*, 1979], and some of the existing systems are still mostly used in this context (see e.g., SB-ONE [Kobsa, 1989]). However, its success in this area has also led to applications in other fields (see e.g., MESON [Edelmann and Owsnicki, 1986] which is used for computer configuration tasks, CLASSIC [Borgida *et al.*, 1989] which is e.g. used in the area of CAD/CAM, or K-REP [Mays *et al.*, 1987; Mays *et al.*, 1988] which is used in a financial marketing domain).

A drawback which pure KL-ONE languages have is that all the terminological knowledge has to be defined on the abstract logical level. In many applications, one would like to be able to refer to concrete domains and predicates on these domains when defining concepts. An example for such a concrete domain could be the set of nonnegative integers. In the above example, one might think that being human and female is not enough to make a woman. As an additional property one could require that she should be old enough, e.g., at least 21. Thus one would like to introduce a new role **age**, and define **Woman** by an expression of the form $\text{Human} \sqcap \text{Female} \sqcap \geq_{21}(\text{age})$. Here \geq_{21} stands for the unary predicate $\{n; n \geq 21\}$ of all nonnegative integers greater or equal 21. Stating such properties directly with reference to a given concrete domain seems to be easier and more natural than encoding them somehow into abstract concept expressions.¹ Though this drawback already appears in natural language processing, it becomes even more important if one has other applications in mind. For example, in a technical application the adequate representation of geometrical concepts requires to relate points in a coordinate system. For that purpose one would e.g. like to have access to real arithmetic. Similar motivations have already led to extensions of KL-ONE in the above mentioned systems MESON, CLASSIC, and K-REP. The MESON system provides “a separate hierarchy for describing non-concepts (e.g., integer ranges and strings)” ([Patel-Schneider *et al.*, 1990], p. 8) which are given as user-defined or machine-defined predicates. Similar features are provided by the “test” construct in CLASSIC. In K-REP “the roles of concepts may in turn be other (complex) concepts, as well as numbers, strings and ... arbitrary Lisp objects” ([Mays *et al.*, 1988], p. 62). Schmiedel’s Temporal Terminological Logic [Schmiedel, 1990] can also be seen in this light. In this case the concrete domain is given by an extension of Allen’s interval calculus [Allen, 1983].

¹See e.g. [Brachman and Schmolze, 1985], Section 9.2, where so-called Structural Descriptions are used to encode the concrete predicate “less than one hour”. From a computational point of view, Structural Descriptions are as bad as Role Value Maps which cause undecidability of subsumption [Schmidt-Schauß, 1989].

For similar reasons, Logic Programming has been extended to Constraint Logic Programming (CLP) (see e.g., [Jaffar *et al.*, 1990; Colmerauer, 1990; Dincbas *et al.*, 1988]). The constraints in CLP languages “state properties directly in the domain of discourse as opposed to having these properties encoded into Prolog terms” ([Lassez, 1987], p. 2).

Before describing our approach for extending a concept language by concrete domains we shall state some of the properties which such an extension should satisfy:

- The extension should still have a formal declarative semantics which is as close as possible to the usual semantics employed for concept languages.
- It should be possible to combine existing inference algorithms for concept languages with well-known reasoning algorithms in the concrete domain in order to get the appropriate algorithms for the extension.
- One should provide a scheme for extending concept languages by various concrete domains rather than constructing a single ad hoc extension for a specific concrete domain. The formal semantics as well as the combination of the algorithms should already be treated on this scheme level.

In order to satisfy these properties it is important to choose an appropriate interface between the concept language and the concrete domain. The interface which we shall use in the present paper was inspired by a construct which is e.g. present in the CLASSIC system, namely coreference constraints (also called agreements) between chains of single-valued roles (also called features).² With such a coreference constraint one can for example express the concept of all women whose father and husband are of the same age by the expression $\text{Woman} \sqcap (\text{father age}) \downarrow (\text{husband age})$. But one cannot express that the husband is even older than the father. This becomes possible if we take the set of nonnegative integers as concrete domain. Then we can simply write $\text{Woman} \sqcap \geq(\text{husband age}, \text{father age})$ where \geq stands for the binary predicate $\{(n, m); n \geq m\}$ on nonnegative integers. More general, our extension will allow to state that feature chains satisfy a (nonnecessarily binary) predicate which is provided by the concrete domain in question.

The next section will contain a formal definition of what we mean by the notion “concrete domain”. In this section we shall also define important properties of such domains, and give examples of concrete domains. Section 3 describes our scheme for extending a concept language by an arbitrary concrete domain. As a starting point for this extension we use the language \mathcal{ALC} of [Schmidt-Schauß and Smolka, 1991]. The reason for choosing this language was that it is large enough to exhibit most of the problems connected with such an extension. Taking a larger language (e.g., including number restrictions) would only mean more work without bringing new insights. Section 4 describes how an assertional component for such an extended concept language can be defined. For both the terminological and the assertional part of our formalism we shall introduce the important inference problems. Section 5 describes an algorithm which can be used to decide all of these problems. As we shall see in Section 6 this algorithm is not only sound but also complete.³ As an example, an instance of the language scheme is used in Section 7 to express Allen’s interval relations [Allen, 1983]. In [Baader, 1991] an extension of \mathcal{ALC} in a

²Agreements on feature chains are just the restriction of Role Value Maps to single-valued (i.e., functional) roles; but unlike Role Value Maps they usually do not cause undecidability of subsumption [Hollunder and Nutt, 1990].

³All the above mentioned systems employ sound but incomplete algorithms.

different direction is presented. It provides role-forming operators including a transitive closure operator. It is shown that the extended language still has a decidable satisfiability problem for concept terms. Section 8 shows that this problem becomes undecidable if concept terms may contain both a transitive closure operator for features and predicates of a concrete domain.

2 Concrete Domains

The following definition formalizes the notion “concrete domain” which has until now only been used in an intuitive sense.

Definition 2.1 *A concrete domain \mathcal{D} consists of a set $dom(\mathcal{D})$, the domain of \mathcal{D} , and a set $pred(\mathcal{D})$, the predicate names of \mathcal{D} . Each predicate name P is associated with an arity n , and an n -ary predicate $P^{\mathcal{D}} \subseteq dom(\mathcal{D})^n$.*

We shall now give some examples of concrete domains.

- In the examples of the introduction we have considered the concrete domain \mathcal{N} which has the set of nonnegative integers as its domain. We have also used the binary predicate name \geq , and one of the unary predicate names \geq_n .
- The concrete domain \mathcal{R} is defined as follows. 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.⁴ 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 $\{(r, s); r \text{ and } s \text{ are real numbers and } r \leq s\}$. Consequently, the predicate associated to the second formula is $\{(r, s); r \text{ and } s \text{ are real numbers}\} = dom(\mathcal{R}) \times dom(\mathcal{R})$.
- The concrete domain \mathcal{Z} is defined as \mathcal{R} with the only difference that $dom(\mathcal{Z})$ is the set of all integers instead of all real numbers.
- Our next example leaves the realm of numbers and arithmetic. Assume that \mathcal{DB} is an arbitrary relational database equipped with an appropriate query language. Then \mathcal{DB} can be seen as a concrete domain where $dom(\mathcal{DB})$ is the set of atomic values in the database. The predicates of \mathcal{DB} are the relations which can be defined over \mathcal{DB} with the help of the query language.
- One can also consider Allen’s interval calculus [Allen, 1983] as concrete domain \mathcal{AL} . Here $dom(\mathcal{AL})$ consists of intervals, and the predicates are built from Allen’s basic interval relations with the help of logical connectives.

⁴For the sake of simplicity we assume here that the formula itself is the predicate name. In applications, the user will probably take his own intuitive names for these predicates.

As mentioned in the introduction, we want to combine inference algorithms for the given concept language with reasoning algorithms for the concrete domain in order to get inference algorithms for the extended concept language. This is only possible if the concrete domain satisfies some additional properties.

For technical reasons we shall have to push negation into concept terms (see Lemma 3.3 below). To make this possible we have to require that the set of predicate names of the concrete domain is *closed under negation*, i.e., if P is an n -ary predicate name in $\text{pred}(\mathcal{D})$ then there has to exist a predicate name Q in $\text{pred}(\mathcal{D})$ such that $Q^{\mathcal{D}} = \text{dom}(\mathcal{D})^n \setminus P^{\mathcal{D}}$. We will usually refer to this predicate name by \overline{P} . In addition, we need a unary predicate name which denotes the predicate $\text{dom}(\mathcal{D})$. The domain \mathcal{N} from above does not satisfy these properties. We should have to add the predicate names $<$, $<_n$. The domains \mathcal{R} , \mathcal{Z} and \mathcal{AL} satisfy the properties. Whether a domain of the form \mathcal{DB} satisfies these properties depends on the query language.

The property which will be formulated now clarifies what kind of reasoning mechanisms are required in the concrete domain. Let P_1, \dots, P_k be k (not necessarily different) predicate names in $\text{pred}(\mathcal{D})$ of arities n_1, \dots, n_k . We consider the conjunction

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

Here $\underline{x}^{(i)}$ stands for an n_i -tuple $(x_1^{(i)}, \dots, x_{n_i}^{(i)})$ 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 $\text{pred}(\mathcal{R})$, and let $P_2(x, y)$ be the predicate $x > y$ in $\text{pred}(\mathcal{R})$. Obviously, neither the conjunction $P_1(x, y) \wedge P_2(x, y)$ nor $P_2(x, x)$ is satisfiable.

Definition 2.2 *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]. However, for the linear case (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]). The concrete domain \mathcal{Z} is not admissible since Hilbert's Tenth Problem—one of the most prominent undecidable problems [Matijević, 1970; Davis, 1973]—is a special case of its satisfiability problem.

Sometimes the adequate modeling of a problem domain could be facilitated if reference to more than one concrete domain would be possible in a terminology. Therefore, we show how two disjoint admissible concrete domains \mathcal{D}_1 and \mathcal{D}_2 can be combined to a new concrete domain $\mathcal{D}_1 \oplus \mathcal{D}_2$. It turns out (Lemma 2.4) that this combination is also admissible.

Definition 2.3 *Assume that \mathcal{D}_1 and \mathcal{D}_2 are admissible concrete domains with predicate names $P_{1,1}, \dots, P_{1,n_1}$ (resp. $P_{2,1}, \dots, P_{2,n_2}$) such that $\text{dom}(\mathcal{D}_1)$ and $\text{dom}(\mathcal{D}_2)$ are disjoint. Then $\mathcal{D}_1 \oplus \mathcal{D}_2$ can be constructed as follows:*

- *The domain $\text{dom}(\mathcal{D}_1 \oplus \mathcal{D}_2)$ is the union of $\text{dom}(\mathcal{D}_1)$ and $\text{dom}(\mathcal{D}_2)$.*

- The predicates of $\mathcal{D}_1 \oplus \mathcal{D}_2$ are

$$\begin{array}{l} Q_{1,1}, \dots, Q_{1,n_1}, \widehat{Q_{1,1}}, \dots, \widehat{Q_{1,n_1}}, \\ Q_{2,1}, \dots, Q_{2,n_2}, \widehat{Q_{2,1}}, \dots, \widehat{Q_{2,n_2}}, \end{array}$$

where the predicates are defined by

- $(x_1, \dots, x_n) \in Q_{\mu,j}$ iff $(x_1, \dots, x_n) \in P_{\mu,j}$, and
- $(x_1, \dots, x_n) \in \widehat{Q_{\mu,j}}$ iff $(x_1, \dots, x_n) \in \overline{P_{\mu,j}}$ or there is an i such that $x_i \in \text{dom}(\mathcal{D}_{\delta(\mu)})^5$.

The reason why $\widehat{Q_{\mu,j}}$ has to be defined this way is that in $\mathcal{D}_1 \oplus \mathcal{D}_2$ the complement has to be considered with respect to $\text{dom}(\mathcal{D}_1 \oplus \mathcal{D}_2) = \text{dom}(\mathcal{D}_1) \cup \text{dom}(\mathcal{D}_2)$ and not just with respect to $\text{dom}(\mathcal{D}_\mu)$.

Lemma 2.4 *If \mathcal{D}_1 and \mathcal{D}_2 are admissible concrete domains, then $\mathcal{D}_1 \oplus \mathcal{D}_2$ is also an admissible concrete domain.*

Proof. We use the same naming conventions as in the previous definition. Obviously, $\mathcal{D}_1 \oplus \mathcal{D}_2$ is a concrete domain. According to Definition 2.2 for $\mu = 1, 2$ there has to be a name $\text{Top}_{\mathcal{D}_\mu}$ for the entire domain of \mathcal{D}_μ . Let $P_{1,1}$ be a name for the empty predicate $\emptyset = \overline{\text{Top}_{\mathcal{D}_1}}$. Then, using the definition of the predicates in $\mathcal{D}_1 \oplus \mathcal{D}_2$, we derive that $\widehat{Q_{1,1}}$ is a name for the domain of $\mathcal{D}_1 \oplus \mathcal{D}_2$. It is also easy to verify $\overline{Q_{\mu,j}} = \widehat{Q_{\mu,j}}$ and $\overline{\widehat{Q_{\mu,j}}} = Q_{\mu,j}$. It remains to be shown that there is a satisfiability test for $\mathcal{D}_1 \oplus \mathcal{D}_2$.

1. Assume that a conjunction

$$\phi = \bigwedge_{i=1}^k P_i(\underline{x}^{(i)})$$

is given, where the P_i are predicates of $\mathcal{D}_1 \oplus \mathcal{D}_2$.

2. Replace in ϕ all occurrence $Q_{\mu,j}(x_1, \dots, x_n)$ by $P_{\mu,j}$ and all occurrences $\widehat{Q_{\mu,j}}(x_1, \dots, x_n)$ by $\overline{P_{\mu,j}} \vee \text{Top}_{\mathcal{D}_{\delta(\mu)}}(x_1) \vee \dots \vee \text{Top}_{\mathcal{D}_{\delta(\mu)}}(x_n)$.
3. Denote the disjunctive normal form of the resulting formula by ϕ' .
4. Let M be a monom in ϕ' .

- (a) If a variable x occurs as an argument of predicates from both domains then M is not satisfiable, because $\text{dom}(\mathcal{D}_1)$ and $\text{dom}(\mathcal{D}_2)$ are disjoint.
- (b) Otherwise, M can be split into conjunctions M_1 and M_2 such that, for $\mu = 1, 2$, M_μ is a conjunction in \mathcal{D}_μ and no variable occurs in both conjunctions. Finally, we observe that M is satisfiable iff the satisfiability tests of the respective admissible concrete domains succeed for M_1 and M_2 respectively.

□

This construction shows that it will not be a restriction that in the next section we shall integrate only one concrete domain into the concept language.

⁵Here and in the following we assume $\delta(1) = 2$ and $\delta(2) = 1$.

3 The Concept Language

We shall now present our scheme for integrating an arbitrary concrete domain \mathcal{D} into the concept language \mathcal{ALC} . The result of this integration will be called $\mathcal{ALC}(\mathcal{D})$.

3.1 Syntax and Semantics

In addition to the usual language constructs of \mathcal{ALC} , the language $\mathcal{ALC}(\mathcal{D})$ allows features (i.e., functional roles) in value restrictions, and predicate names of \mathcal{D} applied to feature chains. For a set F of feature names, a feature chain is just a nonempty word over F .

Definition 3.1 (concept terms and terminologies of $\mathcal{ALC}(\mathcal{D})$)

Let C , R , and F be disjoint sets of concept, role, and feature names. The set of concept terms of $\mathcal{ALC}(\mathcal{D})$ is inductively defined. As a starting point of the induction, any element of C is a concept term (atomic terms). Now let C and D be concept terms, let R be a role name or feature name, $P \in \text{pred}(\mathcal{D})$ be an n -ary predicate name, and u_1, \dots, u_n be feature chains. Then the following expressions are also concept terms:

1. $C \sqcup D$ (disjunction), $C \sqcap D$ (conjunction), and $\neg C$ (negation),
2. $\exists R.C$ (exists-in restriction) and $\forall R.C$ (value restriction),
3. $P(u_1, \dots, u_n)$ (predicate restriction).

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 (i) no concept name appears more than once as a left hand side of a definition, and (ii) \mathcal{T} contains no cyclic definitions.⁶

Please note that the exists-in and the value restrictions are not only defined for roles but also for features. For a feature chain $u = f_1 f_2 \dots f_s$ we shall sometimes use the notations $\exists u.C$ and $\forall u.C$ as abbreviations for $\exists f_1. \exists f_2. \dots \exists f_s. C$ and $\forall f_1. \forall f_2. \dots \forall f_s. C$.

A T-box contains two different kinds of concept names. *Defined concepts* occur on the left hand side of a terminological axiom. The other concepts are called *primitive concepts*. The following is an example of a T-box in $\mathcal{ALC}(\mathcal{N})$. Let **Human**, **Female**, **Mother**, **Woman** be concept names, let **child** be a role name, and let **age** be a feature name. The T-box—which proposes yet another definition of the concept **woman**—consists of the following axioms:

$$\begin{aligned} \text{Mother} &= \text{Human} \sqcap \text{Female} \sqcap \exists \text{child. Human} \\ \text{Woman} &= \text{Human} \sqcap \text{Female} \sqcap (\text{Mother} \sqcup \geq_{21}(\text{age})) \end{aligned}$$

Here **Mother** and **Woman** are defined concepts, and **Human** and **Female** are primitive concepts. The reason for choosing **child** as role and **age** as feature was that an individual can have more than one child, but (s)he has only one age. The next definition gives a model-theoretic semantics for the languages introduced in Definition 3.1.

⁶See [Nebel, 1989; Baader, 1990] for a treatment of cyclic definitions in concept languages.

Definition 3.2 (interpretations and models)

An interpretation \mathcal{I} for $\mathcal{ALC}(\mathcal{D})$ consists of a set $\text{dom}(\mathcal{I})$, the abstract domain of the interpretation, and an interpretation function. The abstract domain and the given concrete domain have to be disjoint, i.e., $\text{dom}(\mathcal{D}) \cap \text{dom}(\mathcal{I}) = \emptyset$. 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 feature name f a partial function $f^{\mathcal{I}}$ from $\text{dom}(\mathcal{I})$ into $\text{dom}(\mathcal{I}) \cup \text{dom}(\mathcal{D})$.

For such a partial function $f^{\mathcal{I}}$ the expression $f^{\mathcal{I}}(x) = y$ is sometimes written as $(x, y) \in f^{\mathcal{I}}$. If $u = f_1 \dots f_n$ is a feature chain, then $u^{\mathcal{I}}$ denotes the composition $f_1^{\mathcal{I}} \circ \dots \circ f_n^{\mathcal{I}}$ of the partial functions $f_1^{\mathcal{I}}, \dots, f_n^{\mathcal{I}}$.⁷

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, let R be a role name or feature name, $P \in \text{pred}(\mathcal{D})$ be an n -ary predicate name, and u_1, \dots, u_n be feature chains. Assume that $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$ are already defined. Then

1. $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, and $(\neg C)^{\mathcal{I}} = \text{dom}(\mathcal{I}) \setminus C^{\mathcal{I}}$,
2. $(\forall R.C)^{\mathcal{I}} = \{x \in \text{dom}(\mathcal{I}); \text{ for all } y \text{ such that } (x, y) \in R^{\mathcal{I}} \text{ we have } y \in C^{\mathcal{I}}\}$ and $(\exists R.C)^{\mathcal{I}} = \{x \in \text{dom}(\mathcal{I}); \text{ there exists } y \text{ such that } (x, y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$,
3. $P(u_1, \dots, u_n)^{\mathcal{I}} = \{x \in \text{dom}(\mathcal{I}); \text{ there exist } r_1, \dots, r_n \in \text{dom}(\mathcal{D}) \text{ such that } u_1^{\mathcal{I}}(x) = r_1, \dots, u_n^{\mathcal{I}}(x) = r_n \text{ and } (r_1, \dots, r_n) \in P^{\mathcal{D}}\}$.

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} .

The philosophy underlying this definition is that we assume that the concrete domain \mathcal{D} is sufficiently structured by the predicates in $\text{pred}(\mathcal{D})$. That means that we do not want to define new classes of elements of $\text{dom}(\mathcal{D})$ or new relations between elements of $\text{dom}(\mathcal{D})$ with the help of our concept language. Consequently, concept terms are always interpreted as subsets of the abstract domain, i.e., an individual of the concrete domain cannot be element of a concept. For this reason, the complement is defined with respect to $\text{dom}(\mathcal{I})$ and not with respect to $\text{dom}(\mathcal{I}) \cup \text{dom}(\mathcal{D})$; roles are only defined on $\text{dom}(\mathcal{I}) \times \text{dom}(\mathcal{I})$; and the features may have values in $\text{dom}(\mathcal{D}) \cup \text{dom}(\mathcal{I})$, but an element of $\text{dom}(\mathcal{D})$ cannot have a feature value.

3.2 Terminological Reasoning

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 classification. The model-theoretic semantics introduced above allows the following formal definition of subsumption. Let \mathcal{T} be a T-box and let A, B be concept names. Then B subsumes A with respect to \mathcal{T} (symbolically $A \sqsubseteq_{\mathcal{T}} B$) iff $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ holds for all models \mathcal{I} of \mathcal{T} .

In our example, it is very easy to see that **Woman** subsumes **Mother**. However, in general it is not at all trivial to determine such relationships. Until recently, sound and complete subsumption algorithms were only known for rather trivial concept languages (see [Levesque and Brachman, 1987]). Consequently, all the existing KL-ONE systems use only sound, but

⁷The composition should be read from left to right, i.e., $f_1^{\mathcal{I}} \circ \dots \circ f_n^{\mathcal{I}}$ means apply first $f_1^{\mathcal{I}}$, then $f_2^{\mathcal{I}}$, and so on.

incomplete algorithms. If such an algorithm gives a positive answer, a subsumption relationship really exists; but if its answer is negative, then we do not know anything. A subsumption relationship may or may not exist. In [Schmidt-Schauß and Smolka, 1991] a sound and complete subsumption algorithm for \mathcal{ALC} is described. The underlying method of constraint propagation was used in [Hollunder *et al.*, 1990] to derive algorithms for various other concept languages. This method can—with appropriate modifications—also be applied to the languages of the form $\mathcal{ALC}(\mathcal{D})$. As a subtask, such an algorithm for $\mathcal{ALC}(\mathcal{D})$ will have to decide satisfiability of conjunctions of the form $\bigwedge_{i=1}^k P_i(\underline{x}^{(i)})$ in the concrete domain. Thus we shall have to require that \mathcal{D} is admissible.

In the literature (e.g., [Levesque and Brachman, 1987; Schmidt-Schauß and Smolka, 1991; Hollunder *et al.*, 1990]), *subsumption* is often defined without reference to a T-box as a relationship *between concept terms*. For two concept terms C, D we say that D subsumes C (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all interpretations \mathcal{I} . Two concept terms C, D are said to be *equivalent* iff C subsumes D and vice versa. Equivalent terms denote the same set in every interpretation.

It is sufficient to find an algorithm which decides subsumption between concept terms since subsumption w.r.t. a T-box can be reduced to this problem; one simply has to unfold the T-box. Unfolding of a T-box means substituting defined concepts which occur on the right hand side of a definition by their defining terms. This process has to be iterated until there remain only primitive concepts on the right hand sides of the definitions. Obviously, this procedure terminates since the terminology is acyclic; and it does not change the meaning of the T-box. In the example, the unfolded definition for **Woman** is $\mathbf{Woman} = \mathbf{Human} \sqcap \mathbf{Female} \sqcap ((\mathbf{Human} \sqcap \mathbf{Female} \sqcap \exists \text{child.Human}) \sqcup_{\geq 21}(\text{age}))$.

Let \mathcal{T} be a T-box, and assume that $A = C$ and $B = D$ are terminological axioms in the unfolded T-box corresponding to \mathcal{T} . Then we have $A \sqsubseteq_{\mathcal{T}} B$ iff $C \sqsubseteq D$.⁸

The subsumption problem for concept terms can now further be reduced to another interesting problem: the satisfiability problem for concept terms. Let C be a concept term. Then C is said to be *satisfiable*⁹ iff there exists an interpretation \mathcal{I} such that $C^{\mathcal{I}} \neq \emptyset$. Thus, an unsatisfiable concept term denotes the empty set in every interpretation, which means that it is worthless.

Obviously, we have $C \sqsubseteq D$ iff $C \sqcap \neg D$ is unsatisfiable. This shows that an algorithm for checking satisfiability of concept terms also yields a subsumption algorithm. The algorithms described in [Schmidt-Schauß and Smolka, 1991] and [Hollunder *et al.*, 1990] are satisfiability algorithms. However, in the present paper we shall not directly give such an algorithm for $\mathcal{ALC}(\mathcal{D})$. Instead we shall reduce the satisfiability problem for concept terms to a problem which will be introduced in the next section: the consistency problem for A-boxes. In Section 5 we shall describe a sound and complete algorithm which decides this problem.

For this algorithm it will be convenient to have all the concept terms in negation normal form. A concept term is in *negation normal form* iff negation signs occur only immediately in front of concept names. If the set of predicate names of the concrete domain \mathcal{D} is closed under negation and contains a name for $\text{dom}(\mathcal{D})$, then any concept term of $\mathcal{ALC}(\mathcal{D})$ can be transformed into an equivalent term in negation normal form by using the transformations

⁸One should however note that the size of the concept terms C, D may be exponential in the size of the original T-box (see [Nebel, 1990]).

⁹Sometimes also called “coherent” or “consistent” in the literature.

described in the following lemma.

Lemma 3.3 *Let \mathcal{D} be a concrete domain such that $\text{pred}(\mathcal{D})$ is closed under negation and contains a name for $\text{dom}(\mathcal{D})$. Assume that this name is $\text{Top}_{\mathcal{D}}$, and let Top be an abbreviation for the concept term $A \sqcup \neg A$ where A is an arbitrary concept name. Let C, D be concept terms of $\mathcal{ALC}(\mathcal{D})$, R be a role name, f be a feature name, P be an n -ary predicate in $\text{pred}(\mathcal{D})$, and u_1, \dots, u_n be feature chains. Then the following transformations preserve equivalence of concept terms:*

1. $\neg(C \sqcup D) \Longrightarrow ((\neg C) \sqcap (\neg D)), \quad \neg(C \sqcap D) \Longrightarrow ((\neg C) \sqcup (\neg D)), \quad \neg(\neg C) \Longrightarrow C$
 $\neg(\forall R.C) \Longrightarrow (\exists R.\neg C), \quad \text{and} \quad \neg(\exists R.C) \Longrightarrow (\forall R.\neg C).$
2. $\neg(\forall f.C) \Longrightarrow ((\exists f.\neg C) \sqcup \text{Top}_{\mathcal{D}}(f)) \quad \text{and} \quad \neg(\exists f.C) \Longrightarrow ((\forall f.\neg C) \sqcup \text{Top}_{\mathcal{D}}(f)).$
3. $\neg P(u_1, \dots, u_n) \Longrightarrow (\overline{P}(u_1, \dots, u_n) \sqcup (\forall u_1.\text{Top}) \sqcup \dots \sqcup (\forall u_n.\text{Top})).$

The first set of transformations is straightforward. The reason why the other transformations are more complex is that features may have values in $\text{dom}(\mathcal{I})$ or $\text{dom}(\mathcal{D})$. For example, an individual a of $\text{dom}(\mathcal{I})$ is in $(\forall f.C)^{\mathcal{I}}$ iff $f^{\mathcal{I}}(a)$ is undefined or $f^{\mathcal{I}}(a) = b$ for an individual b in $C^{\mathcal{I}}$. Since concepts are always interpreted as subsets of $\text{dom}(\mathcal{I})$ this means in particular that $b \notin \text{dom}(\mathcal{D})$. If we negate these conditions we get that $f^{\mathcal{I}}(a)$ has to be defined and that its value must lie in $(\neg C)^{\mathcal{I}}$ or in $\text{dom}(\mathcal{D})$.

4 The Assertional Language

The terminological formalism introduced in the previous section allows to describe knowledge about classes of objects (the concepts) and relationships between these classes (e.g., subsumption relationships which are consequences of the descriptions). Many applications, however, require that one can also say something about objects in the world. For this reason, most KL-ONE systems provide additional assertional capabilities. This assertional part of the system uses the concept terms for making statements about parts of a given world. The expressiveness of this component varies between the rather weak formalism employed in the original KL-ONE system [Brachman and Schmolze, 1985] to full first order predicate logic as used in KRYPTON [Brachman *et al.*, 1985]. We shall now show how to integrate a concrete domain into an assertional language which is similar to the ones used in KANDOR [Patel-Schneider, 1984], MESON [Edelmann and Owsnicki, 1986], CLASSIC [Borgida *et al.*, 1989], or BACK [Nebel and von Luck, 1988].

4.1 Syntax and Semantics

Let \mathcal{D} be an arbitrary concrete domain. We have seen in Section 3 that we have to deal with two different kinds of objects: the individuals of the concrete domain and the individuals in the abstract domain (see Definition 3.2). The names for objects of the concrete domain will come from a set OC of object names, and the names for objects of the abstract domain from a set OA .

Definition 4.1 (assertional axioms and A-boxes for $\mathcal{ALC}(\mathcal{D})$)

Let OC and OA be two disjoint sets of object names. The set of all assertional axioms is defined as follows. Let C be a concept term of $\mathcal{ALC}(\mathcal{D})$, R be a role name, f be a feature name, P be

an n -ary predicate name of \mathcal{D} , and let a, b be elements of \mathbf{OA} and y, y_1, \dots, y_n be elements of \mathbf{OC} . Then the following are assertional axioms:

$$a : C, \quad (a, b) : R, \quad (a, b) : f, \quad (a, y) : f, \quad (y_1, \dots, y_n) : P.$$

An A-box is a finite set of such assertional axioms.

The assertional language can for example be used to express the facts that the woman Lolita, the daughter of Humbert, is married to Vladimir, a man older than Humbert, by the assertional axioms $\text{LOLITA} : \text{Woman}$, $(\text{LOLITA}, \text{HUMBERT}) : \text{father}$, $(\text{LOLITA}, \text{VLADIMIR}) : \text{husband}$, $(\text{HUMBERT}, \text{A1}) : \text{age}$, $(\text{VLADIMIR}, \text{A2}) : \text{age}$, $(\text{A2}, \text{A1}) : >$. Here LOLITA , HUMBERT , and VLADIMIR are elements of \mathbf{OA} , and A1 and A2 are elements of \mathbf{OC} .

It may seem to be a drawback of the above defined assertional language that it disallows the use of specific elements of $\text{dom}(\mathcal{D})$ in the assertions. For example, we are not allowed to write the axiom $(\text{LOLITA}, 12) : \text{age}$. However, if we have a predicate name for the singleton set $\{12\}$, say $=_{12}$, then we can express the same fact by the two axioms $(\text{LOLITA}, \text{A3}) : \text{age}$ and $=_{12}(\text{A3})$. In A-boxes of $\mathcal{ALC}(\mathcal{R})$ one can use algebraic numbers such as $\sqrt{2}$ because the corresponding singleton set $\{\sqrt{2}\}$ corresponds to a predicate name in \mathcal{R} , namely $(x^2 = 2) \wedge x \geq 0$.

Definition 4.2 (interpretations and models)

An interpretation for the assertional language is simply an interpretation for $\mathcal{ALC}(\mathcal{D})$ which, in addition, assigns an object $a^{\mathcal{I}} \in \text{dom}(\mathcal{I})$ to each object name $a \in \mathbf{OA}$, and an object $x^{\mathcal{I}} \in \text{dom}(\mathcal{D})$ to each object name $x \in \mathbf{OC}$. Such an interpretation satisfies an assertional axiom

$$\begin{aligned} a : C \text{ iff } a^{\mathcal{I}} \in C^{\mathcal{I}}, \quad (a, b) : R \text{ iff } (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}, \quad (a, b) : f \text{ iff } f^{\mathcal{I}}(a^{\mathcal{I}}) = b^{\mathcal{I}}, \\ (a, y) : f \text{ iff } f^{\mathcal{I}}(a^{\mathcal{I}}) = y^{\mathcal{I}}, \quad (y_1, \dots, y_n) : P \text{ iff } (y_1^{\mathcal{I}}, \dots, y_n^{\mathcal{I}}) \in P^{\mathcal{D}}. \end{aligned}$$

An interpretation is a model of an A-box \mathcal{A} iff it satisfies all the assertional axioms of \mathcal{A} , and it is a model of an A-box \mathcal{A} together with a T-box \mathcal{T} iff it is a model of \mathcal{T} and a model of \mathcal{A} .

The definition shows that we do not require unique names for the objects.¹⁰ For example, assume that we have the abstract names VLADIMIR and LOLITA'S_FATHER in our A-box. As our knowledge about the world increases, we may learn that Vladimir is in fact Lolita's father. Similarly, if we introduce concrete names A1 , A2 for the ages of two persons PERSON1 , PERSON2 into the A-box, we do not want to assume automatically that these two numbers are different.

Considering an A-box without a corresponding T-box means that all the concepts names occurring in concept terms are assumed to be primitive. For example, if we consider the above A-box concerning Lolita and her family alone, then the concept name Woman in the axiom $\text{LOLITA} : \text{Woman}$ is treated as a primitive concept. However, if we consider it together with the T-box defined in the previous section, then Woman stands for the concept term $\text{Human} \sqcap \text{Female} \sqcap ((\text{Human} \sqcap \text{Female} \sqcap \exists \text{child.Human}) \sqcup \geq_{21}(\text{age}))$.

¹⁰Many KL-ONE based systems have a unique name assumption for their A-box individuals; but for example KL-TWO [Vilain, 1985] does not assume unique names. For our algorithm, it would be easy to handle a unique name assumption for the abstract objects. If we want to treat a unique name assumption for the concrete objects we have to require that the concrete domain contains a predicate name for equality.

4.2 Assertional Reasoning

In the following, \mathcal{A} will always denote an A-box, \mathcal{T} a T-box, C, D concept terms, $a, b \in \text{OA}$ names of abstract objects, and $x, y \in \text{OC}$ names of concrete objects.

An obvious requirement on the represented knowledge is that it should not be contradictory. Otherwise, it would be useless to deduce other facts from this knowledge since logically, everything follows from an inconsistent set of assumptions. However, for a given A-box (or an A-box together with a T-box) it is not necessary to have a model. For example, an A-box containing the axioms $a : C$ and $a : \neg C$, or the axioms $(a, b) : f$, $(a, y) : f$ for a feature name f is contradictory, and thus cannot have a model.

We say that an A-box (an A-box together with a T-box) is *consistent* iff it has a model. Otherwise, it is called *inconsistent*.

For the above mentioned reason it is important to have an algorithm which decides consistency of a given A-box. In addition, it will turn out that such an algorithm can also be used to solve all the other important inference problems, namely subsumption between concepts, satisfiability of concepts, consistency of an A-box together with a T-box, and the so-called instantiation problem.

This last problem is defined as follows. The abstract object a is an *instance* of C with respect to \mathcal{A} (with respect to \mathcal{A} together with \mathcal{T}) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models of \mathcal{A} (for all models of \mathcal{A} together with \mathcal{T}).

As an example, we consider the T-box defining the concepts **Mother** and **Woman** of Section 3, and an A-box containing the axioms (LOLITA, A3) : **age**, =₁₂(A3) and LOLITA : **Woman**. Then LOLITA is an instance of **Mother** with respect to the A-box together with the T-box.

Consistency of—as well as instantiation with respect to—an A-box together with a T-box can easily be reduced to the corresponding problems for A-boxes alone. In fact, one must simply unfold the corresponding T-box, and then replace all defined concept names occurring in concept terms of the A-box by their definitions in the unfolded T-box.

In addition, the instantiation problem can be reduced to the consistency problem as follows: a is an instance of C with respect to \mathcal{A} iff the A-box $\mathcal{A} \cup \{a : \neg C\}$ is inconsistent.

Finally, the satisfiability problem for concept terms (and thus also the subsumption problem) can also be reduced to the consistency problem for A-boxes. In fact, C is satisfiable iff the A-box $\{a : C\}$ is consistent.

5 The Basic Reasoning Algorithm

In this section we shall describe a sound and complete algorithm which decides the consistency of an A-box for $\mathcal{ALC}(\mathcal{D})$, provided that the concrete domain \mathcal{D} is admissible. Such an algorithm for \mathcal{ALC} without concrete domain and features can be found in [Hollunder, 1990]. Since all the inference problems introduced above can be reduced to the consistency problem (see Section 3.2 and 4.2), we thus have

Theorem 5.1 *Let \mathcal{D} be an admissible concrete domain. Then there exists a sound and complete algorithm which is able to decide the following problems for $\mathcal{ALC}(\mathcal{D})$: the subsumption problem w.r.t. a T-box, the subsumption problem and the satisfiability problem for concept terms, the instantiation problem w.r.t. an A-box (w.r.t. an A-box together with a T-box), and the consistency problem for an A-box (an A-box together with a T-box).*

Let \mathcal{A}_0 be an arbitrary A-box for $\mathcal{ALC}(\mathcal{D})$. Without loss of generality we assume that all the concept terms occurring in this A-box are in negation normal form. In principle, the algorithm will start with the given A-box, and transform it with the help of certain rules until one of the following two situations occurs: (i) the obtained A-box is “obviously contradictory”, or (ii) the obtained A-box is “complete”, i.e., one can apply no more rules. In the second case, the complete A-box describes a model of the original A-box.

Because of the presence of disjunction in our language, a given A-box must sometimes be transformed into two different new A-boxes. For that reason, we shall work with sets \mathcal{M} of A-boxes rather than with a single A-box. If we want to test \mathcal{A}_0 for consistency, we start with the singleton set $\mathcal{M}_0 := \{\mathcal{A}_0\}$.

Before we can formulate the transformation rules we need a technical definition. Let \mathcal{A} be an A-box, f be a feature name, a, b, c be names of abstract objects, and x, y be names of concrete objects. If \mathcal{A} contains the axioms $(a, b) : f$ and $(a, c) : f$ (resp. $(a, x) : f$ and $(a, y) : f$) then we call such a pair of axioms a *fork* in \mathcal{A} . Since f is interpreted as a partial function, such a fork means that b and c (resp. x and y) have to be interpreted as the same object. A fork $(a, b) : f, (a, c) : f$ (resp. $(a, x) : f, (a, y) : f$) can be deleted by replacing all occurrences of c in \mathcal{A} by b (resp. all occurrences of y in \mathcal{A} by x).

Definition 5.2 (transformation rules)

Let \mathcal{M} be a finite set of A-boxes, and let \mathcal{A} be an element of \mathcal{M} . The following rules will replace \mathcal{A} by an A-box \mathcal{A}' or by two A-boxes \mathcal{A}' and \mathcal{A}'' . In the formulation of the rules, the letters a, b (possibly with indices) stand for names of abstract objects, and x, y (possibly with indices) stand for names of concrete objects. The letters C, D denote concept terms, the letter R denotes a feature or a role name, the letter P denotes an n -ary predicate name of \mathcal{D} , and the letters u_1, \dots, u_n denote feature chains.

1. The conjunction rule. Assume that $a : (C \sqcap D)$ is in \mathcal{A} and $a : C$ or $a : D$ is not in \mathcal{A} . The A-box \mathcal{A}' is obtained from \mathcal{A} by adding the two axioms $a : C, a : D$ to \mathcal{A} .
2. The disjunction rule. Assume that $a : (C \sqcup D)$ is in \mathcal{A} and neither $a : C$ nor $a : D$ is in \mathcal{A} . The A-box \mathcal{A}' is obtained from \mathcal{A} by adding $a : C$ to \mathcal{A} , and the A-box \mathcal{A}'' is obtained from \mathcal{A} by adding the axiom $a : D$ to \mathcal{A} .
3. The exists-in restriction rule. Assume that $a : \exists R.C$ is in \mathcal{A} and that there is no object name c in OA such that the axioms $(a, c) : R$ and $c : C$ are in \mathcal{A} . Let $b \in \text{OA}$ be a “new” abstract object name (i.e., a name not occurring in \mathcal{A}). First, we add the two axioms $(a, b) : R, b : C$ to \mathcal{A} . If R is a feature name, we may have created a fork by this replacement. If this is the case, we delete this fork as described above. The resulting A-box is the A-box \mathcal{A}' .
4. The value restriction rule. Assume that $a : \forall R.C$ and $(a, b) : R$ are in \mathcal{A} and that $b : C$ is not in \mathcal{A} . The A-box \mathcal{A}' is obtained from \mathcal{A} by adding the axiom $b : C$.
5. The predicate restriction rule. Assume that $a : P(u_1, \dots, u_n)$ is in \mathcal{A} and that the following does not hold:

For the feature chains $u_i = f_{i1} \dots f_{in_i}, i = 1, \dots, n$, there are object names $b_{i1}, \dots, b_{in_i-1} \in \text{OA}$ and $x_i \in \text{OC}$ such that the A-box \mathcal{A} contains axioms $(a, b_{i1}) : f_{i1}, (b_{i1}, b_{i2}) : f_{i2}, \dots, (b_{in_i-1}, x_i) : f_{in_i}$, and $(x_1, \dots, x_n) : P$.

For each of the feature chains $u_i = f_{i1} \dots f_{in_i}$ we choose new object names $b_{i1}, \dots, b_{in_i-1} \in \text{OA}$ and $x_i \in \text{OC}$, and augment the A-box by the axioms $(a, b_{i1}) : f_{i1}, (b_{i1}, b_{i2}) : f_{i2}, \dots, (b_{in_i-1}, x_i) : f_{in_i}$. If we have created new forks, we delete them as described above. Finally we add $(x_1, \dots, x_n) : P$ to obtain the A-box \mathcal{A}' .

We will see in Lemma 6.3 that there cannot be an infinite chain of sets $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \dots$ where each \mathcal{M}_{i+1} is obtained from \mathcal{M}_i by application of one of the above defined rules. Thus if we start with a set $M_1 = \{\mathcal{A}_1\}$ and apply rules as long as possible we finally end up with a complete set M_r , i.e., a set to which no rules are applicable. We shall now formalize what it means that an A-box in this set is “obviously contradictory”.

Definition 5.3 (clash rules)

We use the same name conventions as in Definition 5.2. We say that an A-box \mathcal{A} contains a clash iff one of the following situations occurs in \mathcal{A} :

1. \mathcal{A} contains axioms $(a, x) : f$ and $(a, b) : f$ for a feature name f . This is an obvious contradiction because we should have to identify a concrete object with an abstract object.
2. \mathcal{A} contains axioms $(a, x) : f$ and $a : \forall f.C$. This is an obvious contradiction because a concrete object cannot be an element of a concept.
3. \mathcal{A} contains axioms $a : A$ and $a : \neg A$ for a concept name A . This is an obvious contradiction because an object cannot be both in a set and in its complement.
4. \mathcal{A} contains axioms $(x_1^{(1)}, \dots, x_{n_1}^{(1)}) : P_1, \dots, (x_1^{(k)}, \dots, x_{n_k}^{(k)}) : P_k$, and the corresponding conjunction $\bigwedge_{i=1}^k P_i(\underline{x}^{(i)})$ is not satisfiable in \mathcal{D} . We are able to detect this contradiction because we have assumed that \mathcal{D} is admissible.

Let \mathcal{A}_0 be the A-box which is to be tested for consistency. In a preprocessing step we transform \mathcal{A}_0 into an A-box \mathcal{A}_1 by eliminating all forks. By applying the rules of Definition 5.2 to $M_1 := \{\mathcal{A}_1\}$ as long as possible, we can compute a complete set of A-boxes M_r . Now \mathcal{A}_0 is consistent iff there exists an A-box in M_r which does not contain a clash (see Section 6 for a proof). This characterization yields a decision criterion for consistency of A-boxes because the set M_r is obtained in finitely many steps, and for a given A-box in M_r one can decide whether it contains a clash.

Thus the decision procedure can be defined in a pseudo programming language as follows:

Algorithm 5.4 (consistency test)

The following procedure takes an A-box \mathcal{A}_0 as an argument and checks whether it is consistent or not.

```

define procedure check-consistency( $\mathcal{A}_0$ )
   $\mathcal{A}_1 := \text{eliminate-forks}(\mathcal{A}_0)$ 
   $r := 1$ 
   $M_1 := \{\mathcal{A}_1\}$ 
  while ‘a transformation rule is applicable to  $M_r$ ’ do
     $r := r + 1$ 
     $M_r := \text{apply-a-transformation-rule}(M_{r-1})$ 
  od
  if ‘there is an  $\mathcal{A} \in M_r$  that does not contain a clash’
    then return consistent
  else return inconsistent

```

6 Soundness and Completeness

In this section we shall prove termination, soundness, and completeness of the consistency test (Algorithm 5.4). Taking these facts together, we get that the algorithm is a decision procedure for the consistency of an A-box \mathcal{A}_0 .

Proposition 6.1 *Assume that the algorithm as described in Section 5 is applied to \mathcal{A}_0 . Then*

1. *the algorithm always computes a complete set of A-boxes \mathcal{M}_r in finite time, and*
2. *the initial A-box is inconsistent iff all A-boxes $\mathcal{A} \in \mathcal{M}_r$ contain a clash.*

Proof. The proposition is a consequence of the four lemmata (6.2, 6.3, 6.6, 6.7) stated and proved below. \square

Assume that an A-box \mathcal{B}' has been obtained from an A-box \mathcal{B} by a single fork-elimination step. The first part of the following lemma implies that a model I of \mathcal{B} is also a model of \mathcal{B}' . Conversely, as a consequence of the second part, a model of \mathcal{B}' can always be extended to a model of \mathcal{B} . Hence, fork elimination preserves (in)consistency.

Lemma 6.2 (fork elimination)

Assume that $(a, b) : f$ together with $(a, c) : f$ is a fork.

1. *Then for any interpretation I of an A-box $\mathcal{B} = \mathcal{A} \cup \{(a, b) : f, (a, c) : f\}$ we have $b^I = c^I$.*
2. *Conversely, if there is a model I of an A-box $\mathcal{B}' = \mathcal{A}' \cup \{(a, b) : f\}$ and c is a new object name then I extended by $c^I := b^I$ is a model of $\mathcal{B} = \mathcal{A} \cup \{(a, b) : f, (a, c) : f\}$. Here \mathcal{A} denotes an A-box such that \mathcal{A}' can be obtained from \mathcal{A} by replacing all occurrences of c by b .* \square

In addition, the elimination of finitely many forks in a finite A-box takes finite time. Thus we may without loss of generality assume that we start with a fork free A-box \mathcal{A}_1 .

By the while loop of Algorithm 5.4 the semantic problem of consistency for the A-box \mathcal{A}_1 is reduced to a simple syntactic problem for a finite set \mathcal{M}_r of A-boxes. This syntactic problem is to check whether there is an A-Box in \mathcal{M}_r that does not contain a clash. In order to show the correctness of the reduction, we first have to demonstrate that for all A-boxes \mathcal{A}_1 the loop terminates with a complete set \mathcal{M}_r of A-boxes in finite time.

Assume that a computation using the algorithm is given and that in a single execution of the loop body the A-box \mathcal{A}' has (resp. the A-boxes \mathcal{A}' and \mathcal{A}'' have) been derived by an application of one of the transformation rules to an A-box \mathcal{A} . Then \mathcal{A}' is called a *descendant* (resp. \mathcal{A}' and \mathcal{A}'' are called *descendants*) of \mathcal{A} .

Lemma 6.3 (termination)

The algorithm always computes a complete set of A-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 no infinite sequence of A-boxes $\mathcal{A}_1, \mathcal{A}_2, \dots$ where \mathcal{A}_{i+1} is a descendant of \mathcal{A}_i .

Assume to the contrary that there is such an infinite sequence. We shall map each \mathcal{A}_i 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, we get a contradiction as soon as the following lemma has been established.

Lemma 6.4 *If \mathcal{A}' is a descendant of \mathcal{A} , we have $\Psi(\mathcal{A}) \gg \Psi(\mathcal{A}')$.*

The elements of the set Q will have a rather complex structure. They are finite multisets of 4-tuples. Each component of the tuples is either a finite multiset of nonnegative integers (for the second, third, and fourth component) or a nonnegative integer (for the first component). 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 our 4-tuples are compared with respect to the usual ordering on integers, and the finite multiset components by the multiset ordering induced by this ordering. The whole tuples are ordered lexicographically from left to right, i.e., (c_1, \dots, c_4) is larger than (c'_1, \dots, c'_4) iff there exists $i, 1 \leq i \leq 4$, such that $c_1 = c'_1, \dots, c_{i-1} = c'_{i-1}$, and c_i is larger than c'_i . Since the orderings on the components are well-founded, the lexicographical ordering on the tuples is also well-founded. Finite multisets of these tuples are now compared with respect to the multiset ordering induced by this lexicographical ordering. This is the well-founded ordering \gg on Q mentioned above.

Before we can define the mapping Ψ from A-boxes to elements of Q , we need three more definitions. For two nonnegative integers n, m we denote by $n \dot{-} m$ the asymmetrical difference between n and m , i.e., $n \dot{-} m := n - m$ if $n \geq m$, and $n \dot{-} m := 0$ if $n < m$. For a concept C the size $|C|$ is inductively defined as

1. $|P(u_1, \dots, u_n)| = 1$ for all n -ary predicates of the concrete domain and feature chains u_1, \dots, u_n ,
2. $|Q| = |\neg Q| = 1$ for primitive concepts Q ,
3. $|\forall R.C| = |\exists R.C| = 1 + |C|$ for all value and exists-in restrictions, and
4. $|C \sqcup D| = |C \sqcap D| = |C| + |D|$ for disjunctions and conjunctions.

One difficulty in the termination proof is caused by possible cycles (e. g. $(a, b) : R, (b, c) : S, (c, a) : R$) in the initial A-box. But fortunately, objects introduced during the computation cannot be involved in a cycle. Therefore we distinguish *old objects* that occur already in \mathcal{A}_1 from *new objects* that are introduced during the computation by an exists-in or predicate restriction rule. The definition of the mapping Ψ will assure that the contribution of an old object is always “greater” than the contribution of a new object. To achieve this we shall need the constant M defined as the number of different concept terms C that occur in \mathcal{A}_1 in an axiom or as a subterm. Please observe that this is also the number of different concepts that may occur in the computation, and that M is greater than $\max\{|C|; C \text{ occurs in } \mathcal{A}_1\}$ ¹¹.

We are now ready to define the mapping Ψ . Let \mathcal{A} be an A-box. Then $\Psi(\mathcal{A})$ is the multiset which contains for each object a occurring in \mathcal{A} a 4-tuple $\psi_{\mathcal{A}}(a)$ defined as follows:

¹¹Here and in the following we assume that $\max \emptyset$ is set to 0.

1. Let N be the number of different feature names occurring in \mathcal{A}_1 and for a set S let $\#S$ denote the number of elements of S . The first component of $\psi_{\mathcal{A}}(a)$ is the nonnegative integer

$$\begin{aligned}
& 2M + 1 & - & \#\{a : C; a : C \text{ is in } \mathcal{A}\} \\
+ N & & - & \#\{(a, b) : f; (a, b) : f \text{ is in } \mathcal{A} \text{ and } f \text{ is a feature}\},
\end{aligned}$$

if a is old. Otherwise, it is $\max\{|C|; a : C \text{ is in } \mathcal{A}\}$.

2. The second component of $\psi_{\mathcal{A}}(a)$ is the empty multiset, if a is old. Otherwise, it is the multiset consisting of all positive integers $|C \sqcap D|$ (resp. $|C \sqcup D|$), where $a : C \sqcap D$ (resp. $a : C \sqcup D$) occurs in \mathcal{A} and the conjunction (resp. disjunction) rule is applicable to this assertional axiom.
3. The third component of $\psi_{\mathcal{A}}(a)$ is the multiset consisting of all positive integers $|\exists R.C|$ (resp. $|P(u_1, \dots, u_n)|$), where $a : \exists R.C$ (resp. $a : P(u_1, \dots, u_n)$) occurs in \mathcal{A} and the exists-in (resp. predicate) restriction rule is applicable to this assertional axiom.
4. The fourth component of $\psi_{\mathcal{A}}(a)$ is the multiset consisting of all positive integers $|\forall R.C|$, where $a : \forall R.C$ and $(a, b) : R$ occur in \mathcal{A} and the value restriction rule is applicable to this pair of assertional axioms.

To prove Lemma 6.4 we have to consider the respective 4-tuples corresponding to names of objects that occur in \mathcal{A} or \mathcal{A}' . To reduce the number of cases we introduce the notion of *affected objects* and show that $\psi_{\mathcal{A}}(a)$ is always greater than or equal to $\psi_{\mathcal{A}'}(a)$ for objects that are not affected. Hence, in order to prove the lemma, only affected objects have to be considered.

Without loss of generality, we assume that in the fork elimination steps, due to the application of transformation rules, the newly introduced objects are replaced by the elder ones. We define an object a to be *affected* by the transformation from \mathcal{A} to \mathcal{A}' , if

- the transformation rule has been applied to an axiom $a : C$ in this derivation step, or
- an axiom $(a, b) : R$, $(b, a) : R$, or $a : C$ is in \mathcal{A}' but not in \mathcal{A} .

Please observe that the definition of the third and the fourth component of the 4-tuple of any object as well as the definition of the second component for new objects, have the following structure in common: First a set of (pairs of) axioms is determined. Then this set is made smaller due to the applicability of certain transformation rules. Finally, the remaining set is mapped to a multiset of integers using the $|\cdot|$ -function. Assume that \mathcal{A} is transformed to \mathcal{A}' by a transformation rule. By the definition of transformation rules, non-applicability of a rule to a certain axiom (resp. pair of axioms) comes from the presence of other axioms. Since transformation rules do not remove axioms, we observe that to every axiom (resp. pair of axioms) in \mathcal{A} to which a transformation rule is applicable in the context of \mathcal{A}' , the same rule has already been applicable in the context of \mathcal{A} . Thus there is only one possibility how a derivation step can increase one of the mentioned components for an object name a . A new axiom $a : C$ or $(a, b) : R$ has been asserted.

Similarly, the first component of a 4-tuple of a new object a can only be changed, if a new axiom $a : C$ is asserted. But all this cannot happen for objects that have not been affected.

The fact that transformation rules do not remove axioms also implies that the first component of an old object can only decrease or it remains unchanged in a derivation step.

Hence, we can conclude that in order to show that $\Psi(\mathcal{A}')$ can be obtained from $\Psi(\mathcal{A})$ by replacing 4-tuples by smaller ones, we do not have to worry about 4-tuples of objects which are not affected. The following lemma will be useful for the remaining case of affected objects.

Lemma 6.5 *Assume that a is a new object in \mathcal{A} .*

1. *If there is an outgoing edge¹² from a to b (i.e. an axiom $(a, b) : R$) then b is new.*
2. *There is exactly one object b and one feature or role R such that $(b, a) : R$ is in \mathcal{A} . In other words, new objects have exactly one incoming edge.*
3. *If there is an axiom $(b, a) : R$, the first component of $\psi_{\mathcal{A}}(a)$ is smaller than the first component of $\psi_{\mathcal{A}}(b)$, or both tuples are equal to $(0, \emptyset, \emptyset, \emptyset)$.*

Proof. No transformation rule generates a new incoming edge for an object, beside the case when the object is introduced. Hence, the first part of the lemma is obvious. Taking the same argument and recalling that new objects are introduced along with exactly one incoming edge, we immediately get part two. Now, consider the last claim of the lemma. If there is no axiom $a : C$ then $\psi_{\mathcal{A}}(a)$ is equal to the minimal element $(0, \emptyset, \emptyset, \emptyset)$ and we are done. Otherwise, take an axiom $a : C$ in \mathcal{A} such that $|C|$ is maximal. This axiom must have been introduced applying the value restriction (resp. the exists-in restriction rule) to an axiom $b : \forall R.C$ together with $(b, a) : R$ (resp. $b : \exists R.C$). In both cases we are done because $|\forall R.C| = |\exists R.C| > |C|$. \square

Please recall that we have to show that $\Psi(\mathcal{A}')$ can be obtained from $\Psi(\mathcal{A})$ by replacing some (but at least one) 4-tuple by finitely many smaller ones. We have already seen that the 4-tuples related to not affected objects do not cause any trouble, because they do not increase. Now consider the affected objects.

(1) Assume that the conjunction rule has been applied to $a : C \sqcap D$. The object a is the only object that is affected in this derivation step. If a is old, the first component decreases because at least one of the axioms $a : C$ and $a : D$ is new in \mathcal{A}' . If a is new, the first component of the tuple does not change because of $|C \sqcap D| > |C|$ and $|C \sqcap D| > |D|$. In the second component, $|C \sqcap D|$ is removed and possibly replaced by $|C|$ or/and $|D|$. Hence, it decreases. Starting with $\Psi(\mathcal{A})$ we obtain $\Psi(\mathcal{A}')$ by replacing $\psi_{\mathcal{A}}(a)$ by the strictly smaller tuple $\psi_{\mathcal{A}'}(a)$, and by possibly several other replacements from greater by smaller 4-tuples related to objects not affected.

(2) The disjunction rule can be handled in a similar way.

(3) Assume that the value restriction rule has been applied to $a : \forall R.C, (a, b) : R$. Then a and b are the affected objects. First assume that a and b are equal. Because of part three of Lemma 6.5, this is only possible if a is old. But then the first component already decreases because a new axiom $a : C$ has been asserted. Now assume that a and b are different. We first consider a .

- If a is old, its associated 4-tuple decreases. In fact, the first component cannot increase, as mentioned in the argumentation attached to the affected objects. The second component is always \emptyset . Because a and b are distinct, no new axiom $a : D$ has been asserted, and the third component cannot increase. Finally, the fourth component must decrease, because the value restriction rule is no longer applicable to $a : \forall R.C, (a, b) : R$.

¹²Roles and feature axioms in \mathcal{A} can be visualized by a directed graph where nodes are objects and edges are labeled by features or roles.

- If a is new, its associated 4-tuple also decreases. No new axiom $a : D$ has been asserted, and hence the first and second component remain unchanged. For the third and fourth component the same arguments as in the case of an old a can be used.

We shall now show that $\psi_{\mathcal{A}'}(b)$ is smaller than $\psi_{\mathcal{A}}(b)$ or smaller than $\psi_{\mathcal{A}}(a)$. The latter suffices, because $\psi_{\mathcal{A}}(a)$ is removed from $\Psi(\mathcal{A})$.

- If b is old, its first component already decreases because of the assertion of $b : C$.
- If b is new, we make a case distinction on a .
 - If a is old, the first component of $\psi_{\mathcal{A}'}(b)$ is smaller than the first component of $\psi_{\mathcal{A}}(a)$, because the constant M used in the definition of the first component for old objects is large enough.
 - If a is new, the first component of $\psi_{\mathcal{A}}(a)$ is equal to the first component of $\psi_{\mathcal{A}'}(a)$, which in turn is greater than $\psi_{\mathcal{A}'}(b)$ by part three of Lemma 6.5.

(4) Assume that the application of the exists-in restriction rule to $a : \exists R.C$ yields the new axiom $b : C$. Then a and b are the affected objects. First assume that a and b are equal. Because of part three of Lemma 6.5, this is only possible if a is old. But then the first component already decreases because a new axiom $a : C$ has been asserted. Now assume that a and b are different. We first consider a .

- If a is old, its associated 4-tuple decreases. In fact, the first component cannot increase, as shown above. The second component is always \emptyset . Now, consider the third component. The exists-in restriction rule is no longer applicable to $a : \exists R.C$, and because a and b are distinct, no new axiom $a : D$ has been asserted. Hence, the third component gets smaller.
- The remaining case for a new a is analogous to (3).

The remaining cases related to b correspond to the respective cases in (3).

(5) Assume that the predicate restriction rule has been applied to $a : P(u_1, \dots, u_n)$. We use the same naming as in the definition of the rule. Every affected object is mentioned in the rule as an a , b_{ij} , or an x_i . We observe that

(*) in this derivation step no axiom of the form $c : C$ is asserted.

For old objects, the first component cannot increase, and it may decrease by inserting a new feature axiom. The second component remains \emptyset . For new objects, the first and second component remain unchanged as a consequence of (*). For the object a , the third component strictly decreases, because the predicate-restriction rule is no longer applicable to $a : P(u_1, \dots, u_n)$. For the other affected objects it cannot increase, because of (*). But the fourth component might increase, because there could be a new pair of axioms $b : \forall f.C$, $(b, c) : f$, where the latter has been newly introduced. Fortunately, if b is old, the first component of its 4-tuple must get smaller. If b is new, its first component is already smaller than the first component of $\psi_{\mathcal{A}}(a)$. The latter can be seen along the same lines as in the last part of (3) and (4).

This concludes the proof of Lemma 6.4 and thus the first part of Proposition 6.1. \square

To prove the second part of Proposition 6.1, we now define the notion of *contradictory A-boxes* which is the syntactic equivalent of inconsistent A-boxes. The definition is by induction on the relation “descendant” which we have just proved to be noetherian. An A-box \mathcal{A} occurring in the computation is *contradictory* with respect to a computation iff

- \mathcal{A} does not have descendants and contains a clash, or
- all descendants of \mathcal{A} are contradictory.

Please note that according to this definition \mathcal{A}_1 is contradictory iff after the loop there is no clash free A-box \mathcal{A} in the set M_r .

Lemma 6.6 (soundness)

An A-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 A-box \mathcal{A} is given. We have to show that it does not have a model.

If \mathcal{A} does not have a descendant, it must contain a clash. But obviously, an A-box with a clash cannot have a model. For the induction step, assume to the contrary that \mathcal{A} has a model I . We have to show that the descendant (resp. one of the descendants in case of the disjunction rule) of \mathcal{A} has a model. This will be a contradiction to the induction hypothesis, because all descendants of contradictory A-box are contradictory.

We shall only demonstrate the case of the value restriction rule. The other cases can be treated similarly. Assume that the rule has been applied to the axioms $a : \forall R.C$ and $(a, b) : R$ in \mathcal{A} , generating the descendant \mathcal{A}' . Please, note that \mathcal{A}' is a superset of \mathcal{A} and that the only axiom in \mathcal{A}' that is not in \mathcal{A} is $b : C$. Hence, it suffices to show that I satisfies $b : C$. This is an immediate consequence of the definition of a value restriction. \square

Lemma 6.7 (completeness)

If the initial A-box \mathcal{A}_1 is not contradictory with respect to a given computation then it has a model.

Proof. If \mathcal{A}_1 is not contradictory then there is an A-box $\mathcal{A} \supseteq \mathcal{A}_1$ in \mathcal{M}_r to which none of the clash rules is applicable. We define an interpretation I of \mathcal{A} as follows:

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 axioms of the form $P(x_1, \dots, x_n)$. The interpretation I interprets an $x \in \text{OC}$ as $\alpha(x)$.
2. The domain $\text{dom}(I)$ consists of all the objects of OA occurring in \mathcal{A} .
3. Let Q be a primitive concept name. Then we set $a \in Q^I$ iff $a : Q$ occurs in \mathcal{A} .
4. Let R be a role or feature name. Then we set $(a, b) \in R^I$ iff $(a, b) : R$ occurs in \mathcal{A} . This is well defined even if R is a feature, because there is no fork in \mathcal{A} , and the first clash rule is not applicable.

It will be shown by induction on the size of the axioms that I is not only an interpretation but also a model of \mathcal{A} . Assume that ax is an axiom in \mathcal{A} .

1. Let ax be $(a, b) : R$. Then I satisfies the axiom by definition.

2. Let ax be $a : P(u_1, \dots, u_n)$. We use the same naming conventions as in the predicate restriction rule. Then there is an axiom $P(x_1, \dots, x_n)$ in \mathcal{A} and $u_i^I(a) = x_i^I$ holds for all $i = 1, \dots, n$. Hence by definition of α we get that I satisfies $a : P(u_1, \dots, u_n)$.
3. Let ax be $a : Q$, where Q is a primitive concept name. Then by definition of I we have $a \in Q^I$.
4. Let ax be $a : \neg Q$. Then Q is a primitive concept, and because \mathcal{A} is clash free, $a : Q$ is not in \mathcal{A} . Hence we get by definition of I that ax is satisfied by I .
5. Let ax be $a : C \sqcap D$ (resp. $a : C \sqcup D$). Because no transformation rule is applicable $a : C$ and $a : D$ (resp. $a : C$ or $a : D$) are in \mathcal{A} . By induction $a : C$ and (resp. or) $a : D$ are (resp. is) satisfied by I , and hence ax is satisfied.
6. Let ax be $a : \forall R.C$. If there is an axiom $(a, b) : R$ in \mathcal{A} then b is in \mathbf{OA} , because the second clash rule is not applicable. Hence for $a : \forall R.C$ and all axioms $(a, b) : R$ the value restriction rule has been applied. By induction hypothesis, I satisfies $b : C$ for all these b , and hence ax is satisfied.
7. Let ax be $a : \exists R.C$. Then the exists-in restriction rule has been applied and two axioms $(a, b) : R$ and $b : C$ are in \mathcal{A} . By the induction hypothesis they are satisfied by I , and hence ax is satisfied.

Finally, we use $\mathcal{A}_1 \subseteq \mathcal{A}$ to conclude that I is also a model for \mathcal{A}_1 . □

7 Expressing Interval Relations: An Example

In [Allen, 1983] J. F. Allen proposes a formalism to represent relations between time intervals that is based on 13 disjoint basic relations on pairs of intervals. These relations correspond to a case analysis of the relative positions of the interval borders. In addition, he presents a consistency test for given sets of relations that is built around a transitive closure algorithm. This algorithm uses a big propagation table which has the following form: Given two basic interval relations $c(i_1, i_2)$, $d(i_2, i_3)$ it says which basic relations could possibly apply to (i_1, i_3) .

In this section we show how the instance $\mathcal{ALC}(\mathcal{R})$ of our language scheme can be used to check his choice of basic relations and his propagation table. We will define concepts that correspond to the basic interval relations. The subsumption and the satisfiability test of the T-box can then be used to check that his case analysis of relative positions of interval borders is exhaustive and that there are no overlapping cases. Finally, the consistency test for A-boxes will be used to verify the propagation table.¹³

The set $dom(\mathcal{R})$ together with the predicates $<, \geq, >, \leq, =, \neq$ generates a simple admissible concrete domain that suffices for the purpose of this section. We will write the binary predicates in the more readable infix notation.

An interval i is considered as an ordered pair of real numbers $(x_1, x_2), x_1 \leq x_2$. This is reflected in the definition of the concept **Interval**. Its definition refers to the predicate \leq of the concrete domain, and applies it to the features **left** and **right**.

$$\mathbf{Interval} = (\mathbf{left} \leq \mathbf{right})$$

¹³This example has been implemented by Andreas Abecker and Dennis Drollinger.

Allen's 13 basic relations are binary relations on intervals. Thus, we define a concept **Pair** that groups two intervals using the features **first** and **second**.

$$\text{Pair} = \exists \text{first.Interval} \sqcap \exists \text{second.Interval}$$

Now Allen's 13 basic interval relations can be defined in a straightforward manner as

$$\begin{aligned} \text{Equals} &= \text{Pair} \sqcap (\text{first left} = \text{second left}) \\ &\quad \sqcap (\text{first right} = \text{second right}) \\ \text{Before} &= \text{Pair} \sqcap (\text{first right} < \text{second left}) \\ \text{After} &= \text{Pair} \sqcap (\text{first left} > \text{second right}) \\ \text{Meets} &= \text{Pair} \sqcap (\text{first right} = \text{second left}) \\ \text{Met-by} &= \text{Pair} \sqcap (\text{first left} = \text{second right}) \\ \text{Overlaps} &= \text{Pair} \sqcap (\text{first left} < \text{second left}) \\ &\quad \sqcap (\text{first right} < \text{second right}) \\ &\quad \sqcap (\text{first right} > \text{second left}) \\ \text{Overlapped-by} &= \text{Pair} \sqcap (\text{first right} > \text{second right}) \\ &\quad \sqcap (\text{first left} > \text{second left}) \\ &\quad \sqcap (\text{first left} < \text{second right}) \\ \text{During} &= \text{Pair} \sqcap (\text{first left} > \text{second left}) \\ &\quad \sqcap (\text{first right} < \text{second right}) \\ \text{Contains} &= \text{Pair} \sqcap (\text{first right} > \text{second right}) \\ &\quad \sqcap (\text{first left} < \text{second left}) \\ \text{Starts} &= \text{Pair} \sqcap (\text{first left} = \text{second left}) \\ &\quad \sqcap (\text{first right} < \text{second right}) \\ \text{Started-by} &= \text{Pair} \sqcap (\text{first left} = \text{second left}) \\ &\quad \sqcap (\text{first right} > \text{second right}) \\ \text{Finishes} &= \text{Pair} \sqcap (\text{first left} > \text{second left}) \\ &\quad \sqcap (\text{first right} = \text{second right}) \\ \text{Finished-by} &= \text{Pair} \sqcap (\text{first right} = \text{second right}) \\ &\quad \sqcap (\text{first left} < \text{second left}) \end{aligned}$$

To show that the 13 cases considered by Allen do not overlap, we verify that all pairwise conjunctions of the respective concepts are inconsistent. For example to check that *Meets* does not overlap with *After* we check whether the concept

$$\text{Meets} \sqcap \text{After}$$

is not satisfiable.

It is obvious that the set of all pairs of intervals form a predicate that is more general than each of Allen's interval relations. Nevertheless, we could use the subsumption service to verify, for example, that **Pair** subsumes **Meets**. To see conversely that Allen's case distinction is exhaustive, we show that the disjunction of all corresponding concepts

subsumes **Pair**.

To verify the propagation table we make use of A-box reasoning. Assume that C and D are concepts that correspond to basic interval relations c and d , respectively, and that Pair_j, l_k are object names from OA. Then the A-box \mathcal{A} defined by

$$\begin{aligned} &(\text{Pair}_1, l_1) : \text{first}, & (\text{Pair}_1, l_2) : \text{second}, \\ &(\text{Pair}_2, l_2) : \text{first}, & (\text{Pair}_2, l_3) : \text{second}, \\ &\text{Pair}_1 : C, & \text{Pair}_2 : D, \end{aligned}$$

corresponds to interval relations $c(i_1, i_2)$ and $d(i_2, i_3)$. If we want to know, whether an interval relation e may possibly apply to (i_1, i_3) , then in a first step we extend \mathcal{A} with the following axioms where E denotes the concept corresponding to e .

$$\begin{aligned} &(\text{Pair}_3, i_1) : \text{first}, & (\text{Pair}_3, i_3) : \text{second}, \\ &\text{Pair}_3 : E. \end{aligned}$$

In a second step, the consistency test for A-boxes checks whether the extended A-box has a model. If it has a model M , the basic interval relation $e(i_1, i_3)$ holds in conjunction with $c(i_1, i_2)$ and $d(i_2, i_3)$ for the intervals $i_j = (\text{left}^M(I_j^M), \text{right}^M(I_j^M)), j = 1, 2, 3$. Otherwise such intervals do not exist. Iterating the procedure over all triples of interval relations (c, d, e) we can verify Allen's propagation table.

8 An Undecidability Result

The concept languages we have considered until now do not provide any operators for constructing complex role terms out of role names. However, for many applications it would be convenient if one were allowed to use e.g. transitive closure of roles when defining concepts. For example, assume that we have the concept **Man** and the role **child**. We can easily define the concept **Mos** of all *men* having *only* sons as

$$\text{Mos} = \text{Man} \sqcap \forall \text{child}.\text{Man}.$$

Assume that we also want to express the concept of all *men* having *only* *male* offsprings, for short **Momo**. We cannot just introduce a new role **offspring** because there would be no connection between the two atomic roles **child** and **offspring**. But the intended meaning of **offspring** is that it is the transitive closure of **child**, i.e., in any interpretation \mathcal{I} the binary relation **offspring** ^{\mathcal{I}} should be the transitive closure of the binary relation **child** ^{\mathcal{I}} .

In [Baader, 1991], the following “transitive extension” of the language \mathcal{ALC} is investigated: instead of roles names one may use role terms involving union, composition and transitive closure of roles in concept definitions. In this language, the concept **Momo** can be defined as

$$\text{Momo} = \text{Man} \sqcap \forall \text{trans}(\text{child}).\text{Man}.$$

The semantics of the role operator **trans** is defined in the obvious way, i.e., for any role R and any interpretation \mathcal{I} , one has $\mathbf{trans}(R)^{\mathcal{I}} := \bigcup_{n \geq 1} (R^{\mathcal{I}})^n$. Baader [Baader, 1991] shows that satisfiability and subsumption of concept terms in the extended language are still decidable.

For many applications it is desirable to have both access to an admissible concrete domain and transitive closure of features. As a motivation, consider the example given in Section 7. There we have defined the concept “pair of intervals” as

$$\mathbf{Pair} = \exists \mathbf{first.Interval} \sqcap \exists \mathbf{second.Interval}$$

and for instance the subconcept “pair of successive intervals” as

$$\mathbf{Meets} = \mathbf{Pair} \sqcap (\mathbf{first\ right} = \mathbf{second\ left})$$

To improve the readability we have written the binary concrete predicate “=” in infix notation. Similarly, one can of course define triples, quadruples, etc. of successive intervals; but it is not possible to define the concept “sequence of successive intervals” this way. However, if we were allowed to use transitive closure of features the concept “sequence of intervals” could be defined as

$$\begin{aligned} \mathbf{Sequence} = \exists \mathbf{head.Interval} \sqcap \\ \forall \mathbf{trans(tail)}. (\exists \mathbf{head.Interval}), \end{aligned}$$

where the feature **head** yields the first element of a given sequence and the feature **tail** yields the remaining sequence after the first element is removed. Now the concept “sequence of successive intervals” can be expressed by the concept term

$$\begin{aligned} \mathbf{sequence} \sqcap ((\mathbf{head\ right} = \mathbf{tail\ head\ left}) \sqcup \forall \mathbf{tail.Bottom}) \sqcap \\ \forall \mathbf{trans(tail)}. ((\mathbf{head\ right} = \mathbf{tail\ head\ left}) \sqcup \forall \mathbf{tail.Bottom}) \end{aligned}$$

Here **Bottom** is intended to denote a concept which is always interpreted as the empty set; for example, defining $\mathbf{Bottom} = A \sqcap \neg A$ for an arbitrary concept A would do. Thus the term $\forall \mathbf{tail.Bottom}$ expresses that the feature **tail** is undefined. This means that the end of the sequence is reached.

In the above concept term we have used both transitive closure of features and predicates of a concrete domain. We know that adding one of these two facilities to a concept language such as \mathcal{ALC} leaves the interesting inference problems decidable. However, the situation changes if we want to have both facilities in one language.

If, starting with \mathcal{ALC} , we allow transitive closure of features and integrate the admissible concrete domain \mathcal{R} (which stands for real arithmetic) then the satisfiability problem becomes undecidable.

This will be shown by reducing the Post Correspondence Problem to the satisfiability problem for this language. The reduction will use only very simple predicates from real arithmetic, namely equalities between linear polynomials in at most two variables.

First, we recall the definition of the Post Correspondence Problem. Let Σ be a finite alphabet. A *Post Correspondence System* over Σ is a nonempty finite set $S = \{(l_i, r_i); i = 1, \dots, m\}$ where the l_i, r_i are words over Σ . A nonempty sequence $1 \leq i_1, \dots, i_n \leq m$ is called a *solution* of the system S iff $l_{i_1} \cdots l_{i_m} = r_{i_1} \cdots r_{i_m}$. It is well-known that the *Post Correspondence Problem*,

i.e., the question whether there exists a solution for a given system, is in general undecidable if the alphabet contains at least two symbols [Post,1946].

In order to reduce this problem to a satisfiability problem for concept terms which use transitive closure of features and refer to the concrete domain \mathcal{R} , we have to encode words into real numbers. This can be done as follows. For $B := |\Sigma| + 1$ we can consider the elements of Σ as digits $1, 2, \dots, B - 1$ of numbers represented at base B . For a given nonempty word w over Σ we denote by \overline{w} the nonnegative integer (in ordinary representation at base 10) it represents at base B . We assume that the empty word ε represents the integer 0. Obviously, the mapping $w \mapsto \overline{w}$ is a 1-1-mapping from Σ^* into the set of nonnegative integers. Concatenation of words is reflected on the corresponding numbers as follows. Let v, w be two words over Σ . Then we have $\overline{vw} = \overline{v} \cdot B^{|w|} + \overline{w}$, where $|w|$ denotes the length of the word w .

We are now ready to define names for the predicates of the concrete domain \mathcal{R} we shall use in our reduction. For $i = 1, \dots, m$,

$$\begin{aligned} C_l^i(x, y, z) &\iff y = \overline{l_i} \wedge z = y + x \cdot B^{|l_i|}, \\ C_r^i(x, y, z) &\iff y = \overline{r_i} \wedge z = y + x \cdot B^{|r_i|}, \\ E(x, y) &\iff x = y, \quad \text{and} \quad L(x) \iff x = 0. \end{aligned}$$

Let l, r, w_l, w_r , and f be feature names. The concept term $C(S)$ corresponding to the Post Correspondence System S is now defined as follows:

$$\begin{aligned} C(S) &= \bigsqcup_{i=1}^m \left(C_l^i(w_l, l, f w_l) \sqcap C_r^i(w_r, r, f w_r) \right) \sqcap \\ &\quad L(w_l) \sqcap L(w_r) \sqcap \\ &\quad \forall \text{trans}(f). \left(\bigsqcup_{i=1}^m \left(C_l^i(w_l, l, f w_l) \sqcap C_r^i(w_r, r, f w_r) \right) \right) \sqcap \\ &\quad \exists \text{trans}(f). E(w_l, w_r). \end{aligned}$$

Proposition 8.1 *The concept term $C(S)$ is satisfiable if and only if the Post Correspondence System S has a solution. Consequently, satisfiability is in general undecidable for concept terms which may contain transitive closure of features and predicate restrictions of an admissible concrete domain.*

Proof. Assume that S has a solution i_1, \dots, i_n of length n . We extend this sequence to an infinite sequence $i_1, \dots, i_n, i_{n+1}, i_{n+2}, \dots$ by choosing arbitrary indices $1 \leq i_{n+1}, i_{n+2}, \dots \leq m$. This new sequence is used to define an interpretation \mathcal{I} as follows:

$$\begin{aligned} \text{dom}(\mathcal{I}) &:= \{k; k \geq 1\}, \quad \text{and for all } k \geq 1, \\ f^{\mathcal{I}}(k) &:= k + 1, \\ l^{\mathcal{I}}(k) &:= \overline{l_{i_k}} \quad \text{and} \quad r^{\mathcal{I}}(k) := \overline{r_{i_k}}, \\ w_l^{\mathcal{I}}(k) &:= \overline{l_{i_1} \cdots l_{i_{k-1}}} \quad \text{and} \quad w_r^{\mathcal{I}}(k) := \overline{r_{i_1} \cdots r_{i_{k-1}}}. \end{aligned}$$

Please note that for $k = 1$, the word $l_{i_1} \cdots l_{i_{k-1}}$ is the empty word, and thus $\overline{l_{i_1} \cdots l_{i_{k-1}}} = 0$. It is now easy to show that $1 \in C(S)^{\mathcal{I}}$. Obviously, this implies that $C(S)$ is satisfiable.

On the other hand, assume that $C(S)$ is satisfiable, and let \mathcal{I} be an interpretation such that $C(S)^{\mathcal{I}} \neq \emptyset$. This interpretation can be used to find a solution of S . Consider an arbitrary element c of $C(S)^{\mathcal{I}}$. Obviously, $c \in (L(w_l) \sqcap L(w_r))^{\mathcal{I}}$ yields $w_l^{\mathcal{I}}(c) = 0 = w_r^{\mathcal{I}}(c)$. Since

$$c \in \left(\bigsqcup_{i=1}^m (C_l^i(w_l, l, f w_l) \sqcap C_r^i(w_r, r, f w_r)) \right)^{\mathcal{I}},$$

we know that there exists an index between 1 and m , say i_1 , such that

$$c \in (C_l^{i_1}(w_l, l, f w_l) \sqcap C_r^{i_1}(w_r, r, f w_r))^{\mathcal{I}}.$$

By the definition of the concrete predicates we get that $l^{\mathcal{I}}(c) = \overline{l_{i_1}}$ and $r^{\mathcal{I}}(c) = \overline{r_{i_1}}$, and $(f w_l)^{\mathcal{I}}(c) = \overline{l_{i_1}}$ and $(f w_r)^{\mathcal{I}}(c) = \overline{r_{i_1}}$.

Similarly, one can show by induction on k that for all $k \geq 0$ there exists an index i_{k+1} between 1 and m such that

$$\begin{aligned} (f^k l)^{\mathcal{I}}(c) &= \overline{l_{i_{k+1}}}, & (f^k r)^{\mathcal{I}}(c) &= \overline{r_{i_{k+1}}}, \\ (f^{k+1} w_l)^{\mathcal{I}}(c) &= \overline{l_{i_1} \cdots l_{i_{k+1}}}, & (f^{k+1} w_r)^{\mathcal{I}}(c) &= \overline{r_{i_1} \cdots r_{i_{k+1}}}. \end{aligned}$$

From $c \in (\exists \text{trans}(f).E(w_l, w_r))^{\mathcal{I}}$ we can now deduce that there exists a positive integer n such that $(f^n w_l)^{\mathcal{I}}(c) = (f^n w_r)^{\mathcal{I}}(c)$, and thus we have $\overline{l_{i_1} \cdots l_{i_n}} = \overline{r_{i_1} \cdots r_{i_n}}$. Consequently, $l_{i_1} \cdots l_{i_n} = r_{i_1} \cdots r_{i_n}$, which shows that the sequence i_1, \dots, i_n is a solution of S . \square

9 Conclusion

We have proposed a KL-ONE based knowledge representation and reasoning system which is hybrid in two respects. On the one hand, it makes the usual distinction between two epistemological different kinds of knowledge, the terminological knowledge and the assertional knowledge. On the other hand, the terminological and assertional language, which usually describes the knowledge on an abstract logical level, is extended by allowing to refer to concrete domains and predicates on these domains.

The different parts of the system are integrated with the help of a unified model-theoretic semantics. Reasoning in the terminological and the assertional part can be done with the help of a single basic reasoning algorithm. This algorithm creates subtasks which have to be solved by the special purpose reasoner of the concrete domain (see the fourth clash rule in Definition 5.3). But there is no other interaction necessary between our basic reasoning algorithm and the reasoner on the concrete domain.

Our approach differs from other extensions of KL-ONE which were done for similar reasons in several respects. Firstly, we have proposed a scheme for such an extension, and not a particular extension by some specific concrete domains. The formal semantics and the algorithm are given on this scheme level. Secondly, the basic reasoning algorithm is not only sound but also complete with respect to this semantics. In addition, we can utilize special purpose reasoners which may already exist for the concrete domain in question. This shows another difference to e.g. the MESON system where the important relationships between the user-defined or machine-defined predicates have to be explicitly supplied by the user. Because of the relatively narrow

interface which we allow between the abstract and the concrete part of our formalism, the special purpose reasoner of the concrete domain may be considered as a black box. This is different to e.g. Schmiedel's Temporal Terminological Logic where the terminological and the temporal parts are interleaved in a way which seems to make it impossible to separate the corresponding reasoning components.

Our main motivation for developing the presented KL-ONE extension was to represent knowledge in a mechanical engineering domain. In particular, we wanted to describe both geometric and other attributes of lathe work pieces in a unified framework. For that purpose we intend to use the language $\mathcal{ALC}(\mathcal{R})$ where geometric properties can be described with the help of predicates over real numbers. Unfortunately, linear predicates (i.e., predicates built from equalities and inequalities between linear polynomials) are not sufficient in this context. However, it is possible to specify geometric primitives—such as circle, cone, lateral area of a cone, etc.—which can be used to build up larger predicates. Since these primitives are relatively simple they can be preprocessed with the help of a quantifier elimination procedure for linear statements with parameters (see e.g. [Weispfenning, 1988; Loos and Weispfenning, 1990]). After this elimination—which has to be done only once for a given collection of geometric primitives—the satisfiability problems generated by our basic reasoning algorithm are purely existential problems. These problems can e.g. be solved by the method described in [Canny, 1988].

As already indicated by the examples given in Section 2, there are also other interesting instances of our scheme. For example, by using the language $\mathcal{ALC}(\mathcal{AL})$, or by expressing intervals in $\mathcal{ALC}(\mathcal{R})$ as demonstrated in Section 7, one can get a simple integration of temporal knowledge into KL-ONE. Though this approach is not as expressive as the one of Schmiedel, it may be sufficient for some applications; and it has the obvious advantage that there exist sound and complete reasoning algorithms.

References

- [Allen, 1983] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [Baader and Hanschke, 1991] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991. A long version (the one you are reading now) is available as DFKI Research Report RR-91-10.
- [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. AAAI, 1990. A long version is available as DFKI Research Report RR-90-01.
- [Baader, 1991] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991. A long version is available as DFKI Research Report RR-90-13.
- [Borgida *et al.*, 1989] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A structural data model for objects. In *International Conference on Management*

- [Brachman and Schmolze, 1985] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
- [Brachman *et al.*, 1979] R. J. Brachman, R. J. Bobrow, P. R. Cohen, J. W. Klovstad, B. L. Webber, and W. A. Woods. Research in natural language understanding, annual report. Tech. Rep. No. 4274, Cambridge, MA, 1979. Bolt Beranek and Newman.
- [Brachman *et al.*, 1985] R. J. Brachman, V. Pigman Gilbert, and H. J. Levesque. An essential hybrid reasoning system: knowledge and symbol level accounts in KRYPTON. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 532–539, 1985.
- [Canny, 1988] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 460–467. ACM, 1988.
- [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 of *LNCS*, 1975.
- [Colmerauer, 1990] A. Colmerauer. An introduction to PROLOG III. *Communications of the ACM*, 33(7), 1990.
- [Davis, 1973] M. Davis. Hilbert’s tenth problem is unsolvable. *Am. Math. Monthly*, 80:239–269, 1973.
- [Dershowitz and Manna, 1979] N. Dershowitz and Z. Manna. Proving termination with multi-set orderings. *Communications of the ACM*, 8(22):465–476, 1979.
- [Dincbas *et al.*, 1988] M. Dincbas, P. Van Hentenryck, H. Simonis, and A. Aggoun. The constraint logic programming language CHIP. In *Proceedings of the 2nd International Conference on Fifth Generation Computer Systems*, pages 249–264, 1988.
- [Edelmann and Owsnicki, 1986] J. Edelmann and B. Owsnicki. Data models in knowledge representation systems: a case study. In *GWAI-86 und 2. Österreichische Artificial-Intelligence-Tagung*, volume 124 of *Informatik-Fachberichte*, pages 69–74. Springer, 1986.
- [Hollunder and Nutt, 1990] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. Research Report RR-90-04, DFKI / Kaiserslautern, 1990.
- [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)*, pages 348–353, 1990.
- [Hollunder, 1990] B. Hollunder. Hybrid inferences in KL-ONE-based knowledge representation systems. In *GWAI-90; 14th German Workshop on Artificial Intelligence*, volume 251 of *Informatik-Fachberichte*, pages 38–47. Springer, 1990.
- [Jaffar *et al.*, 1990] J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The CLP(\mathcal{R}) language and system. CMU-CS-90-181, School of Computer Science, Carnegie Mellon University, 1990. Early Version in Proceedings of the 4th International Conference on Logic Programming, May 1987.

- [Kobsa, 1989] A. Kobsa. The SB-ONE knowledge representation workbench. In *Preprints of the Workshop on Formal Aspects of Semantic Networks*, 1989. Two Harbors, Cal.
- [Lassez, 1987] C. Lassez. Constraint logic programming. In *Constraint Logic Programming: A Reader*, 1987. Fourth IEEE Symposium on Logic Programming, San Francisco.
- [Levesque and Brachman, 1987] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 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.
- [Matijacevič, 1970] Y. Matijacevič. Enumerable sets are diophantine. *Soviet Math. Doklady*, 11:354–357, 1970. English translation.
- [Mays *et al.*, 1987] E. Mays, C. Apté, J. Griesmer, and J. Kastner. Organizing knowledge in a complex financial domain. *IEEE Expert*, 2(3):61–70, 1987.
- [Mays *et al.*, 1988] E. Mays, C. Apté, J. Griesmer, and J. Kastner. Experience with K-Rep: an object centered knowledge representation language. In *Proceedings of IEEE CAIA-88*, pages 62–67, 1988.
- [Nebel and von Luck, 1988] B. Nebel and K. von Luck. Hybrid reasoning in BACK. In Z. W. Ras and L. Saitta, editors, *Methodologies for Intelligent Systems*, volume 3, pages 260–269. North-Holland, 1988.
- [Nebel, 1989] B. Nebel. Terminological cycles: Semantics and computational properties. In *Proceedings of the Workshop on Formal Aspects of Semantic Networks*, 1989. Two Harbors, Cal.
- [Nebel, 1990] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2):235–249, 1990.
- [Patel-Schneider *et al.*, 1990] P. F. Patel-Schneider, B. Owsnicki-Klewe, A. Kobsa, N. Guarino, R. McGregor, W. S. Mark, D. McGuinness, B. Nebel, A. Schmiedel, and J. Yen. Report on the workshop on term subsumption languages in knowledge representation. *AI Magazine*, 11(2):16–23, 1990.
- [Patel-Schneider, 1984] P. F. Patel-Schneider. Small can be beautiful in knowledge representation. In *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, pages 11–16. Denver, Colo., 1984. An extended version including a KANDOR system description is available as AI Technical Report No. 37, Palo Alto, Cal., Schlumberger Palo Alto Research, 1984.
- [Post, 1946] E. M. Post. A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.*, 52:264–268, 1946.
- [Schmidt-Schauß and Smolka, 1991] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. To appear in *Journal of Artificial Intelligence*, 47, 1991.

- [Schmidt-Schauß, 1989] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. J. Brachman, editor, *First International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, 1989.
- [Schmiedel, 1990] A. Schmiedel. A temporal terminological logic. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, volume 2, pages 640–645. AAAI, 1990.
- [Tarski, 1951] A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. U. of California Press. Berkley, 1951.
- [Vilain, 1985] M. Vilain. The restricted language architecture of a hybrid representation system. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 547–551, 1985.
- [Weispfenning, 1988] V. Weispfenning. The complexity of linear problems in fields. *J. Symbolic Computation*, 5:3–27, 1988.