Representation and Reasoning with Attributive Descriptions

Bernhard Nebel and Gert Smolka*
IWBS, IBM Deutschland†

Abstract

This paper surveys terminological representation languages and feature-based unification grammars pointing out the similarities and differences between these two families of attributive description formalisms. Emphasis is given to the logical foundations of these formalisms.

1 Introduction

Research in knowledge representation and linguistics has led to the development of two families of formalisms which can jointly be characterized as *attributive description formalisms*. The members of the first family are known as *terminological representation languages* and are offsprings of Brachman's KL-ONE [9], which grew out of research in semantic networks and frame systems. The second family whose members are known as *unification grammars* originated with Kaplan and Bresnan's Lexical-Functional Grammar [17] and Kay's Functional Unification Grammar [19, 21, 20].

This paper surveys terminological representation languages and unification grammars in an attempt to clarify the similarities and differences between the two approaches. Both approaches

1. rely on attributes as the primary notational primitive for representing knowledge
2. are best formalized as first-order logics with Tarski-style models
3. employ compositional set descriptions.

The two approaches differ significantly, however, both in the representational constructs and the reasoning operations they provide. Unification grammars employ functional attributes called *features* while terminological representation languages rely on more general relational attributes called *roles*. The semantically minor-looking difference between features and roles results in very different computational properties. Moreover, terminological representation systems infer set inclusion and set membership relations on user-defined symbols, while parsers based on unification grammars solve constraints in a domain consisting of so-called feature graphs.

Surprisingly, the similarities of terminological representation languages and unification grammars have not been recognized for quite a while. The main reason for this ignorance is probably that

*Funded by the EUREKA Project Protos (EU 56).

†Address for correspondence: IBM Deutschland, IWBS, Postfach 80 08 80, 7000 Stuttgart 80, West Germany; e-mail: nebel@ds0lilog.bitnet, smolka@ds0lilog.bitnet.

the communities of researchers working in the respective fields are almost non-overlapping although recently ideas from feature constraint languages used with unification grammars found their way into terminological representation systems. Nevertheless, so far, a paper putting both approaches into perspective by giving a comprehensible survey of the commonalities and differences is missing—a situation we hope to remedy with this paper.

We emphasize the logical foundations of the two approaches since it is here that the similarities and differences show up most clearly. Moreover, only the development of the logical foundations of terminological representation languages and unification grammars provided the base for the recent surge of important results on and extensions of these formalisms.

The structure of the paper is straightforward. Section 2 discusses terminological representational languages and Section 3 discusses feature-based unification grammars.

Acknowledgement. Our presentation of unification grammars profited from discussions with Jochen Dörre and Bill Rounds.

2 Terminological Representation and Reasoning

One important task in modeling an application domain in artificial intelligence systems is to fix the vocabulary intended to describe the domain—the *terminology*—and to define interrelationships between the atomic parts of the terminology. Representation systems supporting this task are KL-ONE [9] and its descendants [8, 51, 33, 29, 24]. The main idea is to introduce each concept by a *terminological axiom* that associates the new concept with a *concept description* specifying the intended meaning in terms of other concepts.

If we intend to talk, for instance, about persons, we may introduce the concepts Person, Adult, Man, and Woman by relating them to each other as follows:

$$\begin{aligned} \mathsf{Adult} &\sqsubseteq \mathsf{Person} \\ \mathsf{Woman} &\sqsubseteq \mathsf{Adult} \\ \mathsf{Man} &\doteq \mathsf{Adult} \sqcap \neg\mathsf{Woman}. \end{aligned}$$

In other words, Adult is introduced as something specializing Person, Woman as specializing Adult, and men as adults who aren't women.

2.1 The Representational Inventory

Formal *terminologies*, such as the one above, are composed out of *terminological axioms* (TA) which relate a concept (the left hand side) to a concept description (the right hand side) using the *specialization operator* "$\sqsubseteq$" and the *equivalence operator* "$\doteq$":

$$TA \rightarrow A \sqsubseteq C \mid A \doteq C$$

with the additional restriction that no concept may occur more than once as a left hand side in a terminology.

For the moment, let us assume that the right hand sides of terminological axioms—the *concept descriptions* (denoted by C and D)—are composed out of *concepts* (denoted by A and B) and the

following *description-forming operators*:

$$C, D \to A \mid C \sqcap D \mid C \sqcup D \mid \neg C.$$

In order to specify the meaning of terminologies formally, we define an **interpretation** $\mathcal{I}$ as a pair $\langle \mathbf{D}^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ with $\mathbf{D}^{\mathcal{I}}$ an arbitrary set—the *domain*—and $\cdot^{\mathcal{I}}$ a function from concepts to subsets of $\mathbf{D}^{\mathcal{I}}$—the *interpretation function*. Based on that, the interpretation of concept descriptions is defined inductively:

$$\begin{aligned} (C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ (C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\ (\neg C)^{\mathcal{I}} &= \mathbf{D}^{\mathcal{I}} \setminus C^{\mathcal{I}}. \end{aligned}$$

This means that a concept is interpreted as standing for a *set* of objects—its *extension*—and a concept descriptions is interpreted as standing for the set resulting from straightforward applications of set operations corresponding to the description-forming operations. The concepts $\top$ and $\bot$ will be used as abbreviations for $A \sqcup \neg A$ and $A \sqcap \neg A$, respectively, where A is any concept. Thus, $\top$ is interpreted as the set of everything and $\bot$ is interpreted as the empty set.

An interpretation $\mathcal{I}$ **satisfies a terminological axiom** σ, written $\models_{\mathcal{I}} \sigma$, iff the sets denoted by the right hand and left hand side relate to each other as suggested by the symbols:

$$\begin{aligned} \models_{\mathcal{I}} A \doteq D \quad &\textit{iff} \quad A^{\mathcal{I}} = D^{\mathcal{I}} \\ \models_{\mathcal{I}} A \sqsubseteq D \quad &\textit{iff} \quad A^{\mathcal{I}} \subseteq D^{\mathcal{I}}. \end{aligned}$$

Furthermore, an interpretation $\mathcal{I}$ is a **model of a terminology** T, written $\models_{\mathcal{I}} T$, iff all terminological axioms in T are satisfied by $\mathcal{I}$.

Having defined the formal meaning of terminologies, we can now say which *terminological formulas* are *entailed* by a terminology. Here, we will permit arbitrary formulas $C \doteq D$ and $C \sqsubseteq D$. Such a formula τ is **entailed** by a terminology T, written $T \models \tau$, iff τ is satisfied by all models of T. Applying this definition to the introductory example, it is easy to see that the following formulas are entailed:

$$\begin{aligned} \mathsf{Man} &\sqsubseteq \mathsf{Person} \\ \mathsf{Man} \sqcap \mathsf{Woman} &\doteq \bot \\ \mathsf{Man} \sqcup \mathsf{Woman} &\doteq \mathsf{Adult}. \end{aligned}$$

Based on the entailment relation between terminologies and formulas, it is possible to define a relation on the set of concept descriptions, namely, the **subsumption relation** $\preceq_T$ defined as

$$C \preceq_T D \;\; \textit{iff} \;\; T \models C \sqsubseteq D.$$

This relation is obviously a preorder (that is, transitive and reflexive) on the set $\mathit{Cd}(T)$ of concept descriptions composed from symbols appearing in T, and a partial order on the quotient of $\mathit{Cd}(T)$ with respect to the equivalence relation $\approx_T$ defined by

$$C \approx_T D \;\; \textit{iff} \;\; T \models C \doteq D.$$

Although the language defined so far gives a good first impression of the general idea behind terminological representation formalisms, one essential ingredient is missing. All such languages contain constructs to describe concepts by specifying *attributes*—a property which led to the title of this paper.

Reconsidering the introductory example, we note that this terminology contains a certain asymmetry; the meaning of Man is derived from the meaning of Woman but not *vice versa*. In order to remove this asymmetry it is tempting to define these concepts by referring to the sex attribute of a person. Before we can do this, our description language has to be extended, however.

First of all, we need *features* (denoted by f, g and h), which we will interpret as unary partial functions $f^{\mathcal{I}}\colon \mathbf{D}(f^{\mathcal{I}}) \to \mathbf{D}^{\mathcal{I}}$ with $\mathbf{D}(f^{\mathcal{I}}) \subseteq \mathbf{D}^{\mathcal{I}}$. Second, we need *constants* (denoted by a, b, and c), such as male and female, which are interpreted as elements of $\mathbf{D}^{\mathcal{I}}$ under the *unique name assumption*, that is, different constants are assumed to denote different objects:

$$\textit{if } a^{\mathcal{I}} = b^{\mathcal{I}} \textit{ then } a = b.$$

Employing these new syntactic categories, the *description-forming language* is extended to

$$C, D \to \ldots \mid c \mid f\colon C$$

with the interpretation

$$\begin{aligned} c^{\mathcal{I}} &= \{c^{\mathcal{I}}\} \\ (f : C)^{\mathcal{I}} &= \{d \in \mathbf{D}(f^{\mathcal{I}}) \mid f^{\mathcal{I}}(d) \in C^{\mathcal{I}}\}. \end{aligned}$$

Note that our notation is overloaded since a constant c is interpreted as the singleton $\{c^{\mathcal{I}}\}$ if it appears as a concept description.

With this machinery, the introductory example can be rephrased as

$$\begin{aligned} \mathsf{Person} &\sqsubseteq \mathsf{sex}\colon (\mathsf{male} \sqcup \mathsf{female}) \\ \mathsf{Adult} &\sqsubseteq \mathsf{Person} \\ \mathsf{Woman} &\doteq \mathsf{Adult} \sqcap \mathsf{sex}\colon \mathsf{female} \\ \mathsf{Man} &\doteq \mathsf{Adult} \sqcap \mathsf{sex}\colon \mathsf{male}. \end{aligned}$$

Although describing concepts by placing restrictions on features results already in a powerful description language, it is possible to generalize this a bit further. Instead of permitting only single-valued attributes, we may conceive multi-valued attributes. For instance, when we want to talk about the children of a person, we cannot represent them as values of a feature. Accounting for this, we introduce multi-valued attributes, called *roles* and denoted by r, that are interpreted as total functions $r^{\mathcal{I}}\colon \mathbf{D}^{\mathcal{I}} \to 2^{\mathbf{D}^{\mathcal{I}}}$ from elements of $\mathbf{D}^{\mathcal{I}}$ to subsets of $\mathbf{D}^{\mathcal{I}}$. The set $r^{\mathcal{I}}(d)$ will be called the *role-filler set* of d for role r.

Using roles for forming concept description, at least something like "all objects of a role-filler set are of a certain type" and "there exist objects of a certain type" seem to make sense. Thus, let us extend our syntax by *role restrictions*

$$C, D \to \ldots \mid \forall r\colon C \mid \exists r\colon C$$

and assume the interpretations

$$\begin{aligned}(\forall r\colon C)^{\mathcal{I}} &= \{d \in \mathbf{D}^{\mathcal{I}} \mid r^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}\\ (\exists r\colon C)^{\mathcal{I}} &= \{d \in \mathbf{D}^{\mathcal{I}} \mid r^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}.\end{aligned}$$

With this extension of our concept description language[1] we can, for instance, describe the important persons in a family:

$$\begin{aligned}\mathsf{Person} &\sqsubseteq \mathsf{sex}\colon (\mathsf{male} \sqcup \mathsf{female})\\ \mathsf{Parent} &\doteq \mathsf{Person} \sqcap \exists \mathsf{child}\colon \top \sqcap \forall \mathsf{child}\colon \mathsf{Person}\\ \mathsf{Mother} &\doteq \mathsf{Parent} \sqcap \mathsf{sex}\colon \mathsf{female}\\ \mathsf{Grandparent} &\doteq \mathsf{Person} \sqcap \exists \mathsf{child}\colon \mathsf{Parent} \sqcap \forall \mathsf{child}\colon \mathsf{Person}\\ \mathsf{Grandmother} &\doteq \mathsf{Grandparent} \sqcap \mathsf{sex}\colon \mathsf{female}.\end{aligned}$$

Although this terminology looks quite natural, one might ask why we did not mention the role child when introducing the Person concept, that is,

$$\mathsf{Person} \sqsubseteq \mathsf{sex}\colon (\mathsf{male} \sqcup \mathsf{female}) \sqcap \forall \mathsf{child}\colon \mathsf{Person}.$$

Actually, this would have been possible. However, it would have resulted in a **terminological cycle**, a direct (or indirect) occurrence of the concept introduced on the left hand side in the concept description on the right hand side. Such cycles, which are usually not supported in terminological representation systems, are problematical for at least two reasons:

1. the intuitive meaning of a concept containing a terminological cycle is not fully clear
2. it is not straightforward to design subsumption algorithms (see Section 2.2) which deal correctly with terminological cycles.

Concerning the first problem, the semantics of concepts containing cycles, we will note here only that the formal semantics provided so far is adequate to deal with such structures in a satisfying way (see [27, Chapter 5]). A solution for the second problem will be sketched in Section 2.3.

While so far we have talked about how to represent knowledge about concepts by employing terminological axioms, the question comes up: what can you do with your represented knowledge—what are the *computational services* provided by a terminological representation system?

2.2 Computational Services

One service terminological representation systems such as KL-ONE provide, called *classification*, is the computation of a *concept taxonomy*, such as the one in Figure 1, which represents the subsumption relation between concepts for the terminology in the previous subsection. Such a concept taxonomy represents the subsumption relation in a quite dense format. Defining $\trianglelefteq_T$ as a **base** of $\preceq_T$, that is, a *minimal relation* such that the reflexive, transitive closure of $\trianglelefteq_T$ is identical with $\preceq_T$, the relation computed by the classification process can be described as the *base of the subsumption relation on the quotient of the set of concepts with respect to* $\approx_T$. Since the base of a finite partial order is always unique, for every terminology there is a unique concept taxonomy.

[1]Note that the story we told so far is oversimplified. Terminological formalisms used in existing representation systems such as KL-ONE [9], KANDOR [33], BACK [29], CLASSIC [6] use different formalisms. In particular, *features* are used only in CLASSIC. For the purpose of giving the general idea, the current presentation should suffice, though.

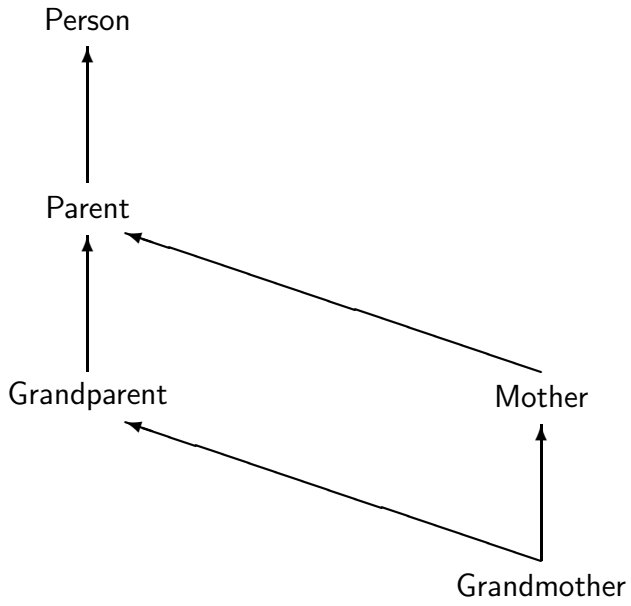

Figure 1: A concept taxonomy.

Evidently, subsumption and classification are intertwined. In order to compute the concept taxonomy, subsumption between concepts must be determined. Once the concept taxonomy has been computed, subsumption between concepts can be read off from the taxonomy,[2] that is, classification can be regarded as a kind of assert-time inference technique.

As can be seen from the example in Figure 1, classification is a non-trivial service. It has to take into account all concept descriptions used in terminological axioms and has to compare them in order to determine the subsumption relationship. For instance, the relationship between Grandparent and Parent, which is not explicitly stated in the terminology, has to be derived by comparing the restrictions on the child role. Although in this example the subsumption relation seems to be quite obvious, the task of determining subsumption can become arbitrarily difficult as we will see in Section 2.3.

Ignoring this unpleasant situation for the moment, we note that classification is a versatile service for the knowledge acquisition task. Classification points out all implicit relationships between concepts which might have been missed when introducing a concept. As shown in [1, 10], classification can be used to drive the knowledge acquisition process by employing a number of reasonable heuristics such as that different concepts should not denote the same set and that no concept should be **incoherent**, that is, equivalent to $\bot$. Note that such incoherent concepts are quite useless because they denote the empty set, but they do not "infect" the knowledge base in the sense a contradictory proposition in logic does. Terminologies always have at least one model, namely, the trivial one interpreting every concept, feature, and role as the empty set.

Proposition 2.1 *Every terminology has a model.*

Knowledge acquisition is not the only application where classification can be put to use. In general, any problem requiring *classification-based reasoning* [15] can exploit this service. This kind

[2]Actually, in some representation systems not only the concept taxonomy but also the transitive closure is stored.

of reasoning proceeds along the following line. Given some concept description, identify the concepts which most accurately characterize the given description and use information associated with the identified concepts to do something meaningful, that is, the concepts are used as a kind of *conceptual coat rack* [52].

Making this idea less abstract, let us assume that we want to identify a plan in order to solve a problem. Now, we may define a hierarchy of problem concepts associating with each such problem concept a plan for solving the problem. Thus, given a particular problem description, classification can determine the most specialized set of problem concepts for which plans are known in order to solve the given problem [30]. Such an organization of problem-solving knowledge is not only very elegant and natural, but also makes maintenance of such a knowledge base easier and supports explanation facilities. Other examples of where this kind of representation and reasoning can be profitably exploited are computer configuration [31], natural language generation [49], presentation planning [4], and information retrieval [50, 35, 5].

However, in most of the applications cited above, one does not start with a description of, say, a particular problem, but one has a collection of *objects* and *relationships* between them. Given such a *world description*,[3] one wants to know the set of concepts most accurately describing those objects. In order to capture this formally, let us again extend our formalism. This time, however, we do not add new description-forming expressions or terminological axioms, but something, which will be called *world axioms* (*WA*) in order to describe objects by naming the concepts they shall be an *instance* of and to describe relationships between two objects by specifying features or roles:

$$WA \rightarrow C(c) \mid f(c,d) \mid r(c,d).$$

Using the interpretation of constants, concepts, roles, and features given above, we will say that a **world axiom is satisfied by an interpretation**, written $\models_{\mathcal{I}} \omega$, under the following conditions:

$$\begin{aligned} \models_{\mathcal{I}} C(c) \quad & \textit{iff} \quad c^{\mathcal{I}} \in C^{\mathcal{I}} \\ \models_{\mathcal{I}} f(c,d) \quad & \textit{iff} \quad f^{\mathcal{I}}(c^{\mathcal{I}}) = d^{\mathcal{I}} \\ \models_{\mathcal{I}} r(c,d) \quad & \textit{iff} \quad d^{\mathcal{I}} \in r^{\mathcal{I}}(c^{\mathcal{I}}). \end{aligned}$$

Similar to the definition of a model of a terminology, we can say what we mean by a **model of a world description** W or by a **model of a world description** W **combined with a terminology** T, namely, an interpretation which satisfies all world axioms in W or all terminological and world axioms in $T \cup W$, respectively. Furthermore, we will say that c is an **instance** of C iff $T \cup W \models C(c)$.

Using this extension of our formalism, we can describe a world specifying the objects and relationships of interest by a world description W employing a terminology T. For instance, we may use the family terminology in order to describe a particular family constellation:

$$\begin{aligned} &\mathsf{Parent}(\mathsf{harry}) \\ &(\mathsf{Parent} \sqcap \mathsf{sex{:}\ female})(\mathsf{mary}) \\ &\mathsf{child}(\mathsf{mary}, \mathsf{tom}) \\ &\mathsf{child}(\mathsf{mary}, \mathsf{harry}). \end{aligned}$$

[3] In the KL-ONE terminology, such a world description is usually called ABox—the *assertional box*—contrasting it with the *terminological box*, the TBox.

From this constellation it follows that mary is a Grandmother and that tom is a Person.

In general, representation systems supporting the reasoning with terminological and world axioms provide a computational service called **realization** which computes for each constant c the set of most specialized concepts $MSC(c)$ the constant is an instance of. Formally, $MSC(c)$ is a minimal set of concepts[4] such that

$$\text{if } A \in MSC(c) \quad \text{then} \quad T \cup W \models A(c) \tag{1}$$

$$\text{if } T \cup W \models B(c) \quad \text{then} \quad \text{there exists } A : A \in MSC(c) \text{ and } T \cup W \models A \sqsubseteq B. \tag{2}$$

In our case, the second condition (2) can be simplified because world descriptions form an (almost) *conservative extension* of terminologies, that is, entailment of terminological formulas depends (in all interesting cases) only on the terminology and not on the world description.

Theorem 2.2 *If a world description W and a terminology T are jointly satisfiable, and if no constant that is used in a terminological axiom occurs in a world axiom, then*

$$T \cup W \models \tau \text{ iff } T \models \tau$$

for all terminological formulas τ.

Proof: The "if" direction is obvious. For the "only if" direction let us assume that there are two descriptions C and D with $T \cup W \models C \sqsubseteq D$ but $T \not\models C \sqsubseteq D$. Since we can extend any model of $T \cup W$ by a disjoint union of a model of T (only agreeing in the interpretation of constants used in terminological axioms), there must be a model of $T \cup W$ which does not satisfy $C \sqsubseteq D$. Thus, our assumption must be wrong. ■

This means that we can formulate condition (2) equivalently as

$$\text{if } T \cup W \models B(c) \quad \text{then} \quad \text{there exists } A : A \in MSC(c) \text{ and } A \preceq_T B,$$

provided the assumptions of the theorem hold. This means in particular that after $MSC(c)$ has been computed, instance relationships for c can be determined by looking up subsumption in the concept taxonomy.[5]

In a presentation planning application (see, for instance, [4]) the information associated with the concepts in MSC might be used to decide how to represent a given object. In a database or information retrieval application, MSC can be used to *index* the data objects by the concepts in MSC [35, 5]. *Query processing* can then be implemented as *classification* of a *query concept*, *retrieval* of all objects indexed by the immediate superconcepts of the query concept in the concept taxonomy, and *filtering* by testing each retrieved object against the query concept.

There are a number of points we have omitted in the presentation of the representational inventory provided by terminological representation systems. In particular, most terminological formalisms allow for a richer repertoire of description-forming operators. First, instead of a simple existential

[4]Actually, of equivalence classes of concepts in order to guarantee uniqueness.

[5]Note that this property heavily depends on the fact that the world axioms have very limited expressiveness. If arbitrary first-order formulas are permitted, as in KRYPTON, the computation of instance relationships becomes much more complicated.

quantification, numerical quantification is usually permitted, for instance, "at least 2 children." Second, roles are not necessarily treated as primitive entities, but they can be defined similarly to the way concepts can be defined. For instance, defining an unspecific subrole of a role or defining a role by restricting the range of another role is a common operation. An example for the latter is the definition of a role son, which can be done by restricting the range of child to sex: male. Third, so-called *role-value-maps*—equality constraints on role-filler sets—are often used, which correspond to *agreements* to be discussed in Section 3.3.

2.3 Algorithmic Considerations and Complexity

Although we have talked about computational services, we haven't given algorithms which do the actual computations. However, instead of specifying inference algorithms (see, e.g. [23, 32, 29, 27]), we will investigate the computational properties of the problems. In particular, it will be shown that terminological reasoning is inherently intractable.

As we have seen in the previous section, subsumption determination is the central operation in a terminological knowledge representation system. This point is reinforced by the fact that all other interesting properties and relations, such as *equivalence of two concepts* ($C \approx_T D$), *incoherency of a concept* ($C \approx_T \bot$), and *disjointness of two concepts* ($(C \sqcap D) \approx_T \bot$) can be reduced to subsumption in linear time.

Proposition 2.3 *Given a terminology T and two concept descriptions C, D:*

1. $C \approx_T \bot$ *iff* $C \preceq_T \bot$
2. $C \approx_T D$ *iff* $C \preceq_T D$ *and* $D \preceq_T C$

Similarly, subsumption can be reduced to incoherency and equivalence.

Proposition 2.4 *Given a terminology T and two concept descriptions C, D:*

1. $C \preceq_T D$ *iff* $(\neg C \sqcap D) \approx_T \bot$.
2. $C \preceq_T D$ *iff* $C \approx_T (C \sqcap D)$.

In other words, when looking for efficient inference algorithms for terminological reasoning systems, we have to find efficient subsumption, equivalence, or incoherency detection algorithms. In order to simplify matters, we will show how subsumption in arbitrary terminologies can be reduced to subsumption in the *empty terminology*, denoted by $\emptyset$. First, we will show that the specialization operator "$\sqsubseteq$" is not essential for the expressiveness of terminological formalisms. Terminologies without this operator will be called **equational** terminologies.

Lemma 2.5 *Any terminology T can be transformed in linear time into a equational terminology T' such that for all $C, D \in Cd(T)$:*

$$C \preceq_T D \quad \textit{iff} \quad C \preceq_{T'} D.$$

Proof: Rewrite each axiom of the form $A \sqsubseteq D$ to $A \doteq \overline{A} \sqcap D$, where $\overline{A}$ is a fresh concept, called *primitive component* of A. Obviously, for any model $\mathcal{I}$ of T there exists a model $\mathcal{I}'$ of T' (setting $\overline{A}^{\mathcal{I}} = A^{\mathcal{I}}$) such that

$$C^{\mathcal{I}} = C^{\mathcal{I}'} \quad \textit{for all } C \in \mathit{Cd}(T)$$

and *vice versa*. Since the interpretation of all concept descriptions is identical, the subsumption relation on $\mathit{Cd}(T)$ is identical. ■

As the second step, a function *Exp* from concept descriptions and equational terminologies to concept descriptions is defined. This function repeatedly replaces all concepts A appearing in a given concept description C by the right hand side of the introduction of A until no further replacements are possible. Thus, $\mathit{Exp}(C, T)$ contains only concepts that do not appear as the left hand side of a terminological axiom in T. This function clearly terminates if T does not contain terminological cycles. Moreover, for every model $\mathcal{I}$ of T

$$C^{\mathcal{I}} = (\mathit{Exp}(C, T))^{\mathcal{I}} \tag{3}$$

since the replaced subexpressions and the replacing expressions in C are identically interpreted in T. From this observation it is almost immediate that subsumption in terminologies can be reduced to subsumption in the empty terminology $\emptyset$.

Theorem 2.6 *Given a equational terminology T and two concept descriptions C, D:*

$$C \preceq_T D \quad \textit{iff} \quad \mathit{Exp}(C, T) \preceq_\emptyset \mathit{Exp}(D, T).$$

Proof: For the "if" direction note that any interpretation is a model of the empty terminology $\emptyset$. Thus, subsumption in $\emptyset$ implies subsumption in any particular terminology. Furthermore, because of (3) the "if" direction holds. The converse direction follows from (3) and the fact that subsumption between concept descriptions of the form $\mathit{Exp}(C, T)$ depend only on interpretations of concepts unconstrained by T. ■

This means when developing inference algorithms, we have to consider only the description-forming part of the formalism. Concentrating on the description-forming language introduced in Section 2.1, which will be called $\mathcal{ALC}$ following [43],[6] it is easy to see that subsumption is decidable for this language.

Lemma 2.7 *Let C and D be $\mathcal{ALC}$ concept descriptions, then it is decidable whether $C \preceq_\emptyset D$.*

Proof Sketch: In order to decide $C \preceq_\emptyset D$, it suffices to check whether $(\neg C \sqcap D)^{\mathcal{I}} = \emptyset$ for all $\mathcal{I}$. Note that if there is a non-empty interpretation of $(\neg C \sqcap D)$, that is, there is an $x \in (\neg C \sqcap D)^{\mathcal{I}}$, then there must be an interpretation of $(\neg C \sqcap D)$ such that there are only n elements y with $f^{\mathcal{I}}(x) = y$ or $y \in r^{\mathcal{I}}(x)$, where n is the number of feature and role restrictions in the top-level expression of $(\neg C \sqcap D)$. Furthermore, this holds also for the (finite number of) concept descriptions embedded in feature and role restrictions. This means, if $(\neg C \sqcap D)$ has a non-empty interpretation at all, then there is also a finite non-empty interpretation bounded in size by $(\neg C \sqcap D)$. Thus, it suffices to check

[6] Actually, in [43] the description-forming language does not contain features. These do not add to the principal complexity of the subsumption problem, however.

only a finite number of finite interpretations in order to decide whether $(\neg C \sqcap D)$ has a necessarily empty interpretation, and, by that, to decide subsumption. ■

Based on this, it is easy to see that subsumption determination for the terminological formalism based on $\mathcal{ALC}$ is decidable—provided there are no terminological cycles in the terminology

Theorem 2.8 *Let T be a cycle-free $\mathcal{ALC}$ terminology and let C, D be concept descriptions. Then it is decidable whether $C \preceq_T D$.*

Proof: Immediate by Lemma 2.5, Theorem 2.6, and Lemma 2.7. ■

Interestingly, decidability is preserved even if terminological cycles are introduced. Although in this case subsumption in a terminology cannot be reduced to subsumption in the empty terminology because *Exp* doesn't terminate, it is possible to define a similar expansion function which expands the concept descriptions only to a finite depth. The main argument for decidability is that the descriptive power of $\mathcal{ALC}$ does not allow to distinguish between infinite interpretations and finite interpretations containing "assertional cycles."[7]

Now there may be the question, what kind of additional description-forming operator would result in the undecidability of subsumption. As has been shown in some recent papers [41, 42, 34], if some very natural looking extensions are added to our language, for instance, equality constraints on role-filler sets or role-negation and role-composition, then subsumption becomes undecidable, even if we consider only the empty terminology.

Despite the positive result concerning decidability, $\mathcal{ALC}$ has an unsatisfying property. Subsumption in the empty terminology is PSPACE-complete as shown in [43]. Since a terminological representation system is supposed to provide its computational services in reasonable time, this is an unacceptable state of affairs. Most terminological representation systems support therefore only less expressive description-forming languages [33] or limit the inference capabilities in a way such that only "interesting" inferences are drawn [26], where "interesting" may be defined by an alternative, weaker set-theoretic semantics [32] or by enumerating (perhaps only implicitly) the possible inference rules.

Following the first suggestion of limiting the expressiveness of the description-forming language, let us analyze some subsets of $\mathcal{ALC}$. First, it should be obvious that any subset containing $\neg$, $\sqcup$, and $\sqcap$ has still an intractable subsumption problem, because the satisfiability problem of propositional logic can be reduced to the problem of determining coherency of a concept description.

Proposition 2.9 *Given a description-forming language that includes $\neg$, $\sqcup$, and $\sqcap$, it is co-NP-hard to decide whether $C \preceq_\emptyset D$.*

For this and other reasons, disjunction is usually banned from terminological formalisms and negation is only permitted on *primitive components* (see proof of Lemma 2.5). However, even with these severe restrictions we do not necessarily achieve tractable subsumption. For the description-forming language containing only $\forall$, $\exists$, $\sqcap$, and $\neg$ on primitive components, it is unknown whether a polynomial subsumption algorithm exists. Only in the case when the second argument of the $\exists$

[7] See [27, Chapter 5] for a decidability proof for a related language.

operator is always $\top$, subsumption is known to be polynomial [23].[8] Although this language, which has been called $\mathcal{FL}^-$ in [7, 23], can be slightly extended without "falling off the computational cliff," the expressiveness is severely restricted if we confine ourselves to description-forming languages that are tractable with respect to subsumption.

Moreover, even if we adopt the point of view that subsumption determination in the empty terminology should be tractable, does that really help? When reducing subsumption in a terminology to subsumption in the empty terminology in Theorem 2.6, nothing was said about time and space bounds of the function *Exp*. As a matter of fact, the application of *Exp* can lead to expressions that are not polynomially bounded in the size of its arguments, as the following example demonstrates:

$$\begin{aligned}
\mathsf{C}_1 &\doteq \forall \mathsf{r}\colon \mathsf{C}_0 \sqcap \forall \mathsf{r}'\colon \mathsf{C}_0 \\
\mathsf{C}_2 &\doteq \forall \mathsf{r}\colon \mathsf{C}_1 \sqcap \forall \mathsf{r}'\colon \mathsf{C}_1 \\
&\vdots \\
\mathsf{C}_n &\doteq \forall \mathsf{r}\colon \mathsf{C}_{n-1} \sqcap \forall \mathsf{r}'\colon \mathsf{C}_{n-1}.
\end{aligned}$$

Here, the size of $\mathit{Exp}(\mathsf{C}_n, T)$ is obviously proportional to 2^n. Of course, better algorithms are conceivable. But as it turns out, the subsumption problem in terminologies cannot be reduced generally to the subsumption problem in the empty terminology in linear time, as the following theorem shows [28]:

Theorem 2.10 *Given a terminological formalism containing only the operators $\doteq$, $\sqcap$, and $\forall$, the problem of deciding whether $C \preceq_T D$ in cycle-free terminologies is co-NP-complete.*

Proof Sketch: The problem of *inequivalence of nondeterministic finite automatons* that generate finite languages, which is known to be NP-complete [11, p. 265], can be reduced to inequivalence of concept descriptions in a cycle-free terminology by a straightforward mapping of states to concepts, input symbols to roles, and transitions to $\forall$ role restrictions.[9] Concepts which correspond to final states of the automatons are labeled with a primitive component. Then two automatons are inequivalent if, and only if, the two concepts corresponding to the respective initial states are inequivalent. For this reason, equivalence determination in terminologies, which is a special case of subsumption, is co-NP-hard. Since nonsubsumption can be easily shown to be in NP, the subsumption problem itself is co-NP-complete. ■

From a theoretical point of view this result means that "the goal of forging a powerful system out of tractable parts" [23, p. 89] cannot be achieved in the area of terminological reasoning.[10] Furthermore, it means that almost all terminological reasoning systems described in the literature that have been conjectured or proven to be tractable with respect to subsumption over the description forming language (in the empty terminology) can be blown up with a carefully thought out example. However, nobody seems to have noticed this fact, and, indeed, terminologies occurring in applications appear to be well-behaved. In this respect and with regard to the structure, our problem is similar to the type inference problem in ML, which seems to be solvable in linear time in all practical

[8] Again, the addition of features would not add to the principal complexity.

[9] One could use features instead without invalidating the argument.

[10] Under the reasonable assumption that we have $\doteq$, $\sqcap$, and $\forall$ restrictions (respectively, feature restrictions) with the standard semantics—which is probably something nobody wants to give up on.

applications encountered so far, but is PSPACE-hard in general [16]. The conclusion one can draw from this strange situation is that although the theory of computational complexity can shade some light on the structure of a problem, one should not be scared by intractability in the first place. It may well be the case that it is possible to find algorithms that are well-behaved in all "normal cases."

3 Unification Grammars

In the last decade a new type of grammar formalisms, now commonly referred to as unification grammars, has evolved from research in linguistics, computational linguistics and artificial intelligence.[11] In contrast to augmented transition networks, one of their precursors, unification grammar formalisms provide for the declarative or logical specification of linguistic knowledge. Nevertheless, unification grammars are aimed towards operational use in parsing and generating natural language.

Like any grammar formalism, unification grammars define sets of sentences. But in addition, a unification grammar assigns to every grammatical sentence one or several so-called feature graphs representing syntactic and semantic information. For instance, the syntactic structure of the sentence

John sings a song

can be represented by the feature graph in Figure 2. This graph states that the sentence consists of a subject (John), a predicate (sings) and an object (a song). It also states that the agent of the singing is given by the subject of the sentence and what is sung is given by the object of the sentence. Moreover, the graph states that the tense of the sentence is present.

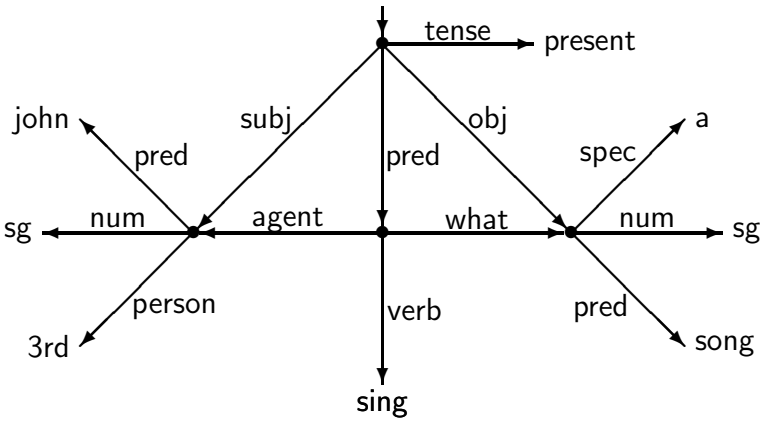

Figure 2: A feature graph.

The linguistic knowledge represented by a unification grammar can be used for parsing and generating sentences in natural language. Suppose "$U(S, G)$" is the relation between strings of symbols and feature graphs specified by some grammar U. Then a string of symbols S is a *sentence of* U if and only if there exists at least one feature graph G such that $U(S, G)$ holds. A parser computes for a given string of symbols S the set $\{G \mid U(S, G)\}$ of all feature graphs related to S,

[11] For the sake of a short name, we use the term unification grammars to stand for feature-based unification grammars, which excludes term-based unification grammars such as Definite Clause Grammars [37].

and a generator computes for a given feature graph G the set $\{S \mid U(S, G)\}$ of all sentences related to G.

Unification grammars are written in some formalism providing for the declarative specification of grammar relations "$U(S, G)$". Ideally, a unification grammar formalism should be general enough to adopt easily to different linguistic theories. Since unification grammars are supposed to be declarative specifications, seeing a unification grammar formalism as a logic for specifying grammar relations makes good sense.

Obviously, there is a strong analogy between logic programs and unification grammars. A logic program is a specification in Horn clause logic that can be employed operationally in various ways. In fact, Definite Clause Grammars [37], which are specified as Horn clauses (which are one possible form of definite clauses), are an early form of unification grammars lacking the notion of features and feature graphs, which evolved independently in (computational) linguistics. So what we need to capture unification grammars as logical specifications is an integration of features and feature graphs with definite clauses. This can be nicely accomplished with the new model of logic programming called Constraint Logic Programming (CLP) [13, 12]. In contrast to Kowalski's [22] classical Horn clause model, CLP is parameterized with respect to a general notion of constraint language. Thus, in order to capture unification grammars as logical specifications, one needs a suitable constraint language for describing feature graphs. Feature constraint languages are in fact the essence of unification grammar formalisms.

The thing feature-based unification grammars have in common with the terminological axioms discussed in the previous section is that both formalisms rely mainly on attributes for representing knowledge. However, the descriptions used and the reasoning services supported are quite different. In KL-ONE-like systems two kinds of attributes—features and roles—are used and the primary reasoning services are determining inclusion between concepts and membership of constants in concepts. Descriptions in unification grammars employ only features and the primary reasoning operation is constructing the most general feature graphs satisfying a description.

The outline of this section is as follows. We start with a formal definition of feature graphs and then devise a suitable constraint language for partially describing feature graphs. Then we show how unification grammars can be written as definite clauses over this constraint language, discuss possible extensions of the basic model, and give pointers to the literature. Finally, we prove the most important properties of our constraint language and give a constraint solving algorithm.

Our view of unification grammars as definite clauses over feature constraint languages is by no means standard and, to our knowledge, has not appeared explicitly in the literature. Our presentation is often informal and relies on previous work on feature logic [48] and definite relations over constraint languages [12].

3.1 A Simple Feature Logic

We assume that an infinite set of **variables** (denoted by x, y, z), a set of **features** (denoted by f, g, h), and a set of **constants** (denoted by a, b, c) are given.

A feature graph is a finite, rooted, connected and directed graph whose edges are labeled with feature symbols such that the labels of the edges departing from a node are pairwise distinct. Moreover, every inner node of a feature graph is a variable and every terminal node is either a constant or a variable.

Formally, an **f-edge from x to s** is a triple xfs such that x is a variable, f is a feature, and s is either a variable or a constant. A **feature graph** is either a pair $(a, \emptyset)$, where a is a constant and $\emptyset$ is the empty set, or a pair (x_0, E), where x_0 is a variable (the **root**) and E is a finite, possibly empty set of edges such that

1. the graph is determinate, that is, if $xfs \in E$ and $xft \in E$, then $s = t$
2. the graph is connected, that is, if $xfs \in E$, then E contains edges leading from the root x_0 to the node x.

To obtain the right notion of feature graph, we will identify all feature graphs that are equal up to consistent variable renaming.

A **feature algebra** is a pair $(\mathbf{D}^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a nonempty set $\mathbf{D}^{\mathcal{I}}$ (the **domain** of $\mathcal{I}$) and an **interpretation function** $\cdot^{\mathcal{I}}$ assigning to ever constant a an element $a^{\mathcal{I}} \in \mathbf{D}^{\mathcal{I}}$ and to every feature f a unary partial function $f^{\mathcal{I}}$ from $\mathbf{D}^{\mathcal{I}}$ to $\mathbf{D}^{\mathcal{I}}$ such that the following conditions are satisfied:

1. if $a \neq b$, then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ *(unique name assumption)*
2. for no feature f and no constant a, the partial function $f^{\mathcal{I}}$ is defined on $a^{\mathcal{I}}$ *(constants are primitive)*.

Note that feature algebras are interpretations in the sense of the previous section, where there are no roles and concepts, and no feature is defined on a constant. Feature algebras can also be seen as Tarski interpretations of the predicate calculus, if we view features equivalently as binary predicates that must be interpreted as functional relations.

There are three reasons for introducing feature algebras. First, we will see that the set of all feature graphs can be regarded naturally as a feature algebra. Second, the constraint language we are going to develop can be interpreted over every feature algebra and we will see that the algebra of all feature graphs takes a prominent position with respect to this constraint language. And third, this approach makes explicit the close connection between Predicate Logic and the particular feature logic we are going to develop.

The **feature graph algebra** $\mathcal{F}$ is obtained as follows:

1. $\mathbf{D}^{\mathcal{F}}$ is the set of all feature graphs
2. $a^{\mathcal{F}}$ is the feature graph $(a, \emptyset)$
3. $f^{\mathcal{F}}$ is defined on a feature graph if and only if there is an f-edge departing from its root
4. if $G = (x, E)$ and $xfs \in E$, then $f^{\mathcal{F}}(G)$ is the largest feature graph (s, E') such that $E' \subseteq E$.

One verifies easily that $\mathcal{F}$ is a feature algebra.

Next we define the constraint language. From features, constants and variables we obtain **terms** as in Predicate Logic. To ease our notation, we omit parentheses and write fs for $f(s)$. A **feature equation** is a pair $s \doteq t$ consisting of two terms. Feature equations are the only primitive constraints we need.

Let $\mathcal{I}$ be a feature algebra. As in Predicate Logic, an **$\mathcal{I}$-assignment** is a mapping from the set of all variables to the domain of $\mathcal{I}$. The interpretation of terms in $\mathcal{I}$ under an $\mathcal{I}$-assignment α are defined as one would expect:

$$\begin{aligned} \mathcal{I}[\![x]\!]_\alpha &= \alpha(x) \\ \mathcal{I}[\![a]\!]_\alpha &= a^{\mathcal{I}} \\ \mathcal{I}[\![fs]\!]_\alpha &= f^{\mathcal{I}}(\mathcal{I}[\![s]\!]_\alpha). \end{aligned}$$

Note that the interpretation of a term fs is not always defined since features are interpreted as partial functions. In particular, no term having a subterm fa has an interpretation. A **solution** of an equation $s \doteq t$ in $\mathcal{I}$ is an $\mathcal{I}$-assignment α such that $\mathcal{I}[\![s]\!]_\alpha$ and $\mathcal{I}[\![t]\!]_\alpha$ are defined and $\mathcal{I}[\![s]\!]_\alpha = \mathcal{I}[\![t]\!]_\alpha$.

A **feature clause** is a finite, possibly empty set of feature equations representing their conjunction. Consequently, a **solution** of a feature clause C in a feature algebra $\mathcal{I}$ is an $\mathcal{I}$-assignment that solves every feature equation in C. If $\mathcal{I}$ is a feature algebra, C is a feature clause and α is a solution of C in $\mathcal{I}$, then we call $\alpha(x)$ a **solution of x in C and $\mathcal{I}$**.

One verifies easily that the feature graph in Figure 2 is a solution of x in the feature graph algebra $\mathcal{F}$ and the feature clause C consisting of the equations

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} \mathsf{pred\ subj\ x} & \doteq & \mathsf{john} & \mathsf{agent\ pred\ x} & \doteq & \mathsf{subj\ x} & \mathsf{spec\ obj\ x} & \doteq & \mathsf{a} \\ \mathsf{num\ subj\ x} & \doteq & \mathsf{sg} & \mathsf{verb\ pred\ x} & \doteq & \mathsf{sing} & \mathsf{num\ obj\ x} & \doteq & \mathsf{sg} \\ \mathsf{person\ subj\ x} & \doteq & \mathsf{3rd} & \mathsf{what\ pred\ x} & \doteq & \mathsf{obj\ x} & \mathsf{pred\ obj\ x} & \doteq & \mathsf{song} \end{array}$$

$$\mathsf{tense\ x} \doteq \mathsf{present}.$$

This clause admits infinitely many other solutions for x that are obtained by adding further edges to the inner nodes of the feature graph in Figure 2. However, the graph in Figure 2 is the most general solution of x in C in that it realizes exactly what is required by the clause and nothing else.

To capture the idea of being more general formally, we define a partial order on feature graphs usually called subsumption.[12] A **morphism** is a function that maps every variable to a variable or a constant, and that maps every constant to itself. If (s, E) and (s', E') are feature graphs, we write $(s, E) \leq (s', E')$ and say that (s', E') is **more specific** than (s, E) or conversely that (s, E) is **more general** than (s', E') if and only if there exists a morphism γ such that $\gamma(s) = s'$ and $\gamma(E) \subseteq E'$, where $\gamma(E) = \{\gamma(x) f \gamma(t) \mid xft \in E\}$. Since we identify feature graphs that are equal up to consistent variable renaming, this definition yields in fact a partial order on the set of all feature graphs.

Proposition 3.1 *For every feature graph G and every variable x one can compute in linear time a feature clause C such that G is the most general solution of x in C and $\mathcal{F}$.*

Proof: If $G = (a, \emptyset)$, then $\{x \doteq a\}$ is a clause as claimed. If $G = (x, E)$, then $\{fy \doteq s \mid yfs \in E\}$ is a clause as claimed. Otherwise, if some variable other than x is the root node of G, x can be made the root node by consistent variable renaming. ■

[12] Note that the subsumption relation on feature graphs defined here is different from the subsumption relation on concept descriptions defined in Section 2.

The most important properties of our constraint language are stated by the following theorem, which we will prove in Subsection 3.4.

Theorem 3.2 *Let C be a feature clause, x be a variable, and $\mathcal{F}$ be the feature graph algebra. Then the following conditions are equivalent:*

1. *C has a solution in some feature algebra*
2. *C has a solution in $\mathcal{F}$*
3. *x has a most general solution in C and $\mathcal{F}$.*

Furthermore, there is a quadratic time algorithm that, given a clause C and a variable x, either returns fail if C has no solution or returns the most general feature graph solution of x in C.

3.2 Unification Grammars as Definite Clauses over Feature Logic

We will now outline a simple unification grammar formalism in which a grammar is given as a set of definite clauses over the feature logic just presented. Such definite clauses can be seen as context-free syntax rules constrained with feature equations.

Figure 3 shows how ordinary context-free grammars translate into Horn clauses. This translation, which is well-known from Definite Clause Grammars [37], assumes that a string of symbols $w_1, \ldots, w_n$ is represented as the term $w_1.\cdots.w_n.\mathsf{nil}$, where the dot is a binary function symbol written in right-associative infix notation. A string of symbols S is a sentence of the grammar if and only if the statement $\mathsf{s}(S)$ follows logically from the Horn clause translation of the grammar.

S	$\longrightarrow$	NP VP	$\mathsf{s(X)} \leftarrow \mathsf{np(X,Y)} \wedge \mathsf{vp(Y,nil)}$
VP	$\longrightarrow$	V NP	$\mathsf{vp(X,Y)} \leftarrow \mathsf{v(X,Z)} \wedge \mathsf{np(Z,Y)}$
NP	$\longrightarrow$	D N	$\mathsf{np(X,Y)} \leftarrow \mathsf{d(X,Z)} \wedge \mathsf{n(Z,Y)}$
NP	$\longrightarrow$	john	$\mathsf{np(john.X,X)} \leftarrow$
V	$\longrightarrow$	sings	$\mathsf{v(sings.X,X)} \leftarrow$
D	$\longrightarrow$	a	$\mathsf{d(a.X,X)} \leftarrow$
N	$\longrightarrow$	song.	$\mathsf{n(song.X,X)} \leftarrow .$

Figure 3: A context-free grammar and its translation into Horn clauses.

Given the Horn clause translation of a context-free grammar, we can introduce for every phrase predicate an additional argument that ranges over feature graphs. Furthermore, in the body of every clause constraints for these additional feature graph arguments can be given. For instance, consider the clauses

$$\mathsf{np(X,Y,NP)} \leftarrow \mathsf{d(X,Z,D)} \wedge \mathsf{n(Z,Y,N)} \wedge \mathsf{NP} \doteq \mathsf{D} \wedge \mathsf{D} \doteq \mathsf{N}$$
$$\mathsf{d(a.X,X,D)} \leftarrow \mathsf{spec(D)} \doteq \mathsf{a} \wedge \mathsf{num(D)} \doteq \mathsf{sg}$$
$$\mathsf{n(song.X,X,N)} \leftarrow \mathsf{pred(N)} \doteq \mathsf{song} \wedge \mathsf{num(N)} \doteq \mathsf{sg},$$

which may be written in the following more intelligible syntax:

$$\mathsf{NP} \longrightarrow \mathsf{D\ N} \qquad \mathsf{NP} \doteq \mathsf{D} \doteq \mathsf{N}$$

$$\mathsf{D} \longrightarrow \mathsf{a} \qquad \mathsf{spec\ D} \doteq \mathsf{a} \wedge \mathsf{num\ D} \doteq \mathsf{sg}$$

$$\mathsf{N} \longrightarrow \mathsf{song} \qquad \mathsf{pred\ N} \doteq \mathsf{song} \wedge \mathsf{num\ N} \doteq \mathsf{sg}.$$

These clauses already illustrate one possible way unification grammars can enforce agreement between the numerus of a noun and its determiner. Since the determiner and the noun are forced by the first clause to carry the same feature graphs (which must also be the feature graphs of the entire noun phrase), it suffices if the rules for the determiner and the noun constrain the feature num independently. In case they constrain the feature num with conflicting values, there exists no feature graph satisfying both constraints.

Saying that a unification grammar consists of definite clauses over a feature constraint language is a little bit oversimplified. Our rules actually employ a three-sorted constraint language providing the sort of all feature graphs, the sort of all words, and the sort of all strings of words. It is straightforward to accommodate this technically. It is also possible to have a single-sorted approach if one codes strings of words as feature graphs.

Figure 4 shows a unification grammar that covers our example sentence "John sings a song". The grammar is given in a sugared syntax that can be translated automatically into definite clauses.[13]

$$\mathsf{S} \longrightarrow \mathsf{NP\ VP} \qquad \mathsf{S} \doteq \mathsf{VP} \wedge \mathsf{subj\ S} \doteq \mathsf{NP}$$

$$\mathsf{VP} \longrightarrow \mathsf{V\ NP} \qquad \mathsf{VP} \doteq \mathsf{V} \wedge \mathsf{obj\ VP} \doteq \mathsf{NP}$$

$$\mathsf{NP} \longrightarrow \mathsf{D\ N} \qquad \mathsf{NP} \doteq \mathsf{D} \doteq \mathsf{N}$$

$$\mathsf{NP} \longrightarrow \mathsf{john} \qquad \mathsf{pred\ NP} \doteq \mathsf{john} \wedge \mathsf{num\ NP} \doteq \mathsf{sg} \wedge \mathsf{person\ NP} \doteq \mathsf{3rd}$$

$$\mathsf{V} \longrightarrow \mathsf{sings} \qquad \mathsf{tense\ V} \doteq \mathsf{present} \wedge \mathsf{verb\ pred\ V} \doteq \mathsf{sing} \wedge \mathsf{agent\ pred\ V} \doteq \mathsf{subj\ V} \wedge \mathsf{what\ pred\ V} \doteq \mathsf{obj\ V} \wedge \mathsf{num\ subj\ V} \doteq \mathsf{sg} \wedge \mathsf{person\ subj\ V} \doteq \mathsf{3rd}$$

$$\mathsf{D} \longrightarrow \mathsf{a} \qquad \mathsf{spec\ D} \doteq \mathsf{a} \wedge \mathsf{num\ D} \doteq \mathsf{sg}$$

$$\mathsf{N} \longrightarrow \mathsf{song} \qquad \mathsf{pred\ N} \doteq \mathsf{song} \wedge \mathsf{num\ N} \doteq \mathsf{sg}$$

Figure 4: A unification grammar.

Since our unification grammars are definite clauses over feature logic, they enjoy a logical semantics provided by the constraint logic programming model [12]. Let x be a fixed variable and let, for every feature graph G, $C[x, G]$ denote a feature clause such that G is the unique most general solution of x in $C[x, G]$ and $\mathcal{F}$. Then a grammar U with the sentence predicate s_U defines a relation "$U(S, G)$" between strings of symbols and feature graphs as follows:

$$U(S,G) \iff U \models C[x,G] \to s_U(S,x) \ \wedge \ \forall G' \colon\ U \models C[x,G'] \to s_U(S,x) \ \Rightarrow\ G \leq G',$$

[13] An equation, say, agent pred V $\doteq$ subj V in our term-oriented syntax would be written in PATR's [47] syntax as $<$ V pred agent $>$ $=$ $<$ V subj $>$.

where $U \models C[x, G] \rightarrow s_U(S, x)$ means that the implication $C[x, G] \rightarrow s_U(S, x)$ follows logically from U and $G \leq G'$ means that G' is more specific than G.

It is possible to verify that the unification grammar in Figure 4 relates the sentence "John sings a song" to the feature graph in Figure 2 and to no other feature graph.

The word problem of a grammar U is to decide for a given string S of symbols whether there exists a feature graph G such that $U(S, G)$ holds. One can show that our formalism allows for grammars having an undecidable word problem by adapting proofs given by Johnson [14] and Rounds and Manaster-Ramer [40] for slightly different formalisms.

Every grammar U in our formalism comes with a context-free grammar CF[U] such that U is obtained from CF[U] by adding feature equations to the rules of U. A grammar U satisfies the *off-line parsability constraint* [17] if the number of different derivations of a string of symbols in CF[U] is bounded by a computable function of the length of that string. Using the operational semantics of definite clauses [12], it is easy to see that for a grammar U satisfying the off-line parsability constraint the word problem is decidable, and that, for every string of symbols S, the set $\{G \mid U(S, G)\}$ is finite and can be computed from S. The off-line parsability constraint is, for instance, satisfied if the right-hand side of every rule of CF[U] contains either at least one terminal or at least two nonterminals.

3.3 Background and Extensions

The simple unification grammar formalism sketched here bears much resemblance with the PATR formalism developed at SRI International by Stuart Shieber and his colleagues [47, 44]. It is also closely related to Kaplan and Bresnan's Lexical-Functional Grammar formalism (LFG) [17]. LFG and PATR were conceived and developed at a time when the constraint logic programming model was not available and have been described quite differently by their inventors. In fact, even the most recent attempts at formalizing unification grammar formalisms [14, 46] still don't make use of the constraint logic programming model.

Shieber's [45] introduction to unification-based approaches to grammar is an excellent survey of existing formalisms and provides the linguistic motivations our presentation is lacking. Other state of the art guides into this fascinating area of research are [38] and [36]. Johnson's thesis [14] gives a formal account of LFG and investigates a feature constraint language with disjunctions and negations. Furthermore, Shieber's [46] thesis gives a rigorous formalization of the PATR formalism.

Why are unification grammars called "unification" grammars? In this context unification is understood as the operation that, given two feature graphs G_1 and G_2, decides whether there exists a feature graph that is more specific than both G_1 and G_2 and, if so, returns the most general such feature graph (which then in fact uniquely exists). Feature graph unification can be seen as an operation combining the information given by two feature graphs provided it doesn't conflict. Due to the close relation between clauses and feature graphs, feature graph unification can be employed as the central operation of a parser emulating a unification grammar, an implementation technique that is elaborated carefully in [46]. Nevertheless, the name "unification grammar", which can be traced back to Martin Kay's [19, 21, 20] Functional Unification Grammar, is rather misleading since it is derived from an operation that may or may not be employed in implementations of these grammar formalisms. Incidentally, in Kaplan and Bresnan's [17] clear and insightful presentation of LFG feature graph unification is not even mentioned.

Kay's Functional Unification Grammar (FUG) [19, 21, 20] is more general than our formalism

since it is not based on context-free rules but uses more flexible mechanisms for establishing word order. Rounds and Manaster-Ramer [40] present a logical formalization of FUG.

The operational semantics of the constraint logic programming model [13, 12] given by goal reduction results in a top-down parsing strategy when applied to our unification grammar formalism. The role of term unification in ordinary Horn clause programming is taken by a constraint solver that simplifies sets of feature equations and thereby checks their solvability. Such a constraint solver generalizes feature graph unification. Existing unification grammar systems often employ chart parsing techniques realizing a bottom up strategy.

The feature constraint language presented here restricts constraints to conjunctions of equations. One obvious extension is to admit other logical connectives such as disjunction, negation or implication. For instance, a lexical rule for the verb "sing" may come with an implicational constraint:

$$\mathsf{V} \longrightarrow \mathsf{sing}$$
$$(\mathsf{person\ subj\ V} \doteq \mathsf{3rd} \ \rightarrow\ \mathsf{num\ subj\ V} \doteq \mathsf{pl}) \ \wedge\ \ldots\ .$$

Deciding for such general constraints whether they have a solution is an NP-complete problem [14, 48]. Furthermore, such a general constraint can have more than one most general feature graph solution, but at most finitely many. Feature constraint languages with all propositional connectives have been investigated by Johnson [14] and Smolka [48].

The integration of Prolog-like logic programming with feature constraint languages seems to be a very promising line of research. Concrete language proposals based on this idea are LOGIN [3] and CIL [25]. The theoretical foundations for this kind of languages have been established in [48, 13, 12].

Kasper and Rounds [18, 39] were the first to develop a constraint logic for feature graphs. Their logic accounts for concept descriptions interpreted in the feature graph algebra $\mathcal{F}$. Their concept descriptions, which we call **feature terms**, are given by the abstract syntax rule

$$S, T \ \longrightarrow\ a \mid p \downarrow q \mid f\!:\! S \mid S \sqcap T \mid S \sqcup T.$$

The new construct $p \downarrow q$, which we call **agreement**, consists of two strings p and q of features (called **paths**). Given a feature algebra $\mathcal{I}$, the interpretation $p^{\mathcal{I}}$ of a string of features $p = f_n \cdots f_1$ is the partial function obtained as the composition of the partial functions $f_n^{\mathcal{I}}, \ldots, f_1^{\mathcal{I}}$, where $f_1^{\mathcal{I}}$ is applied first. The empty string denotes the identity function of $\mathbf{D}^{\mathcal{I}}$. Now the interpretation of an agreement $p \downarrow q$ in $\mathcal{I}$ is the greatest subset of $\mathbf{D}^{\mathcal{I}}$ on which $p^{\mathcal{I}}$ and $q^{\mathcal{I}}$ agree, that is,

$$(p \downarrow q)^{\mathcal{I}} = \{d \in \mathbf{D}(p^{\mathcal{I}}) \cap \mathbf{D}(q^{\mathcal{I}}) \mid p^{\mathcal{I}}(d) = q^{I}(d)\},$$

where $\mathbf{D}(p^{\mathcal{I}})$ and $\mathbf{D}(q^{\mathcal{I}})$ are the domains of the partial functions $p^{\mathcal{I}}$ and $q^{\mathcal{I}}$, respectively. Agreements are needed for expressing *coreferences* in feature graphs, that is, the fact that two paths lead to the same node. Interpreted in the feature graph algebra $\mathcal{F}$, every feature term denotes a set of feature graphs.

Figure 5 gives an example of a feature term written in matrix notation. The feature terms given as the rows of a matrix are connected by intersection. The feature graph in Figure 2 is the most general element of the set of feature graphs denoted by the feature term in Figure 5 in the feature graph algebra $\mathcal{F}$.

Agreements generalize to roles and are known as role value maps in KL-ONE [9]. A recent paper of Schmidt-Schauß [42] shows that role value maps result in an undecidable subsumption relation on

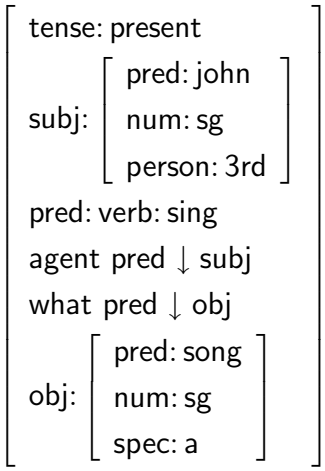

Figure 5: A feature term in matrix notation.

concept descriptions. This is in sharp contrast to agreements for feature terms, which don't cause a blow-up of the computational complexity.

Feature terms can be used for constraining variables if the logic provides for memberships $x\colon S$ consisting of a variable and a feature term, where an $\mathcal{I}$-assignment α is a solution of $x\colon S$ in a feature algebra $\mathcal{I}$ if and only if $\alpha(x) \in S^{\mathcal{I}}$. One can show that memberships not employing unions have the same expressivity as conjunctions of feature equations. Smolka [48] investigates various feature term languages and shows how they reduce to equational constraint languages.

Unification grammars usually have most of their structural information in their lexical rules. Since lexica for realistic subsets of natural language are large, techniques are needed for expressing lexical generalizations so as to allow lexical entries to be written in a compact notation. One such technique is the use of so-called templates in the PATR formalism, which turn out to be noncyclic equational terminological axioms based on feature terms. Figure 6 gives an example.

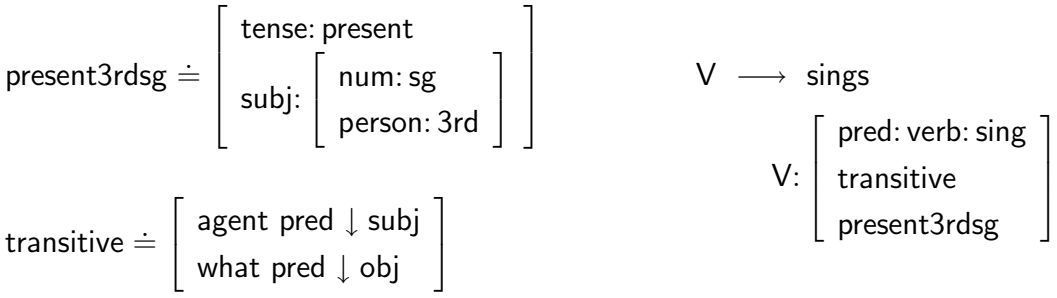

Figure 6: A lexical entry using templates.

3.4 Solving Feature Clauses

We now prove Theorem 3.2 by exhibiting a quadratic-time algorithm for solving feature clauses. The algorithm is given abstractly as a collection of simplification rules providing for the solution-preserving transformation of feature clauses to a solved form.[14]

[14]Theorem 3.2 and the algorithm are taken from [48].

We use $\mathcal{I}[\![C]\!]$ to denote the set of all solutions of a feature clause C in a feature algebra $\mathcal{I}$. Two feature clauses C and D are called **equivalent** if $\mathcal{I}[\![C]\!] = \mathcal{I}[\![D]\!]$ for every feature algebra $\mathcal{I}$. Let V be a set of variables. Then two feature clauses C and D are called V**-equivalent** if for every feature algebra $\mathcal{I}$ the following two conditions are satisfied:

1. if $\alpha \in \mathcal{I}[\![C]\!]$, then there exists $\beta \in \mathcal{I}[\![D]\!]$ such that α and β agree on V
2. if $\alpha \in \mathcal{I}[\![D]\!]$, then there exists $\beta \in \mathcal{I}[\![C]\!]$ such that α and β agree on V.

Proposition 3.3 *Let C and D be V-equivalent feature clauses, $x \in V$ and $\mathcal{I}$ be a feature algebra. Then $d \in \mathbf{D}^{\mathcal{I}}$ is a solution of x in C and $\mathcal{I}$ if and only if d is a solution of x in D and $\mathcal{I}$.*

Our solution algorithm for feature clauses consists of a linear-time unfolding phase followed by a quadratic-time normalization phase.

A feature clause is **unfolded** if each of its equations has either the form $s \doteq t$ or $fs \doteq t$, where s and t range over variables and constants. In other words, a feature clause is unfolded if each of its equations contains either no feature or exactly one feature that occurs on its left-hand side.

Let C be a feature clause. Then we use

1. $\mathcal{V}C$ to denote the set of all variables occurring in C
2. $[x/s]C$ to denote the clause that is obtained from C by replacing every occurrence of the variable x with the term s
3. $s \doteq t \,\&\, C$ to denote the feature clause $\{s \doteq t\} \cup C$ provided $s \doteq t \notin C$.

Unfolding is done by iteratively replacing a term fs in an unfolded position with a fresh variable x and adding the constraint $fs \doteq x$. For instance, the clause $\{fgx \doteq hy\}$ can be unfolded to

$$\{fz \doteq u, \quad gx \doteq z, \quad hy \doteq u\}$$

by introducing the new variables u and z. Unfolding steps are justified by the following proposition:

Proposition 3.4 *Let D be a feature clause, $fs \doteq x \in D$, $C = [x/fs](D - \{fs \doteq x\})$, and let x not occur in fs. Then C and D are $\mathcal{V}C$-equivalent.*

Proposition 3.5 *For every feature clause C one can compute in linear time a $\mathcal{V}C$-equivalent unfolded feature clause D.*

Next we present the solved form for feature clauses to which the normalization phase of our solution algorithm attempts to transform unfolded feature clauses.

A feature clause C is **solved** if it satisfies the following conditions:

1. every equation in C has one of the following forms: $x \doteq y$, $x \doteq a$, $fx \doteq y$ or $fx \doteq a$
2. if $fx \doteq s$ and $fx \doteq t$ are in C, then $s = t$
3. if $x \doteq s$ is in C, then x occurs only once in C.

Let C be a solved feature clause. Then $x \to_C y \iff \exists\, fx \doteq y \in C$ defines a binary relation $\to_C$ on the variables occurring in C. We use $\to_C^*$ to denote the reflexive and transitive closure of $\to_C$ on the set of all variables. If x is a variable, then

$$\mathrm{FG}[x, C] := \begin{cases} (a, \emptyset) & \text{if } x \doteq a \in C \\ \mathrm{FG}[y, C] & \text{if } x \doteq y \in C \\ (x,\ \{yfs \mid fy \doteq s \in C \ \wedge\ x \to_C^* y\}) & \text{otherwise} \end{cases}$$

defines a feature graph.

Theorem 3.6 *If C is a solved feature clause and x is a variable, then* $\mathrm{FG}[x, C]$ *is the most general solution of x in C and $\mathcal{F}$.*

The proof of this theorem is straightforward.

Next we present the normalization phase of the algorithm, which is given by the following solution-preserving simplification rules for unfolded feature clauses:

1. $s \doteq s \ \&\ C \ \to\ C$
2. $x \doteq s \ \&\ C \ \to\ x \doteq s \ \&\ [x/s]C$ if $x \in \mathcal{V}C$ and $x \neq s$
3. $s \doteq x \ \&\ C \ \to\ x \doteq s \ \&\ C$ if s is not a variable
4. $fx \doteq s \ \&\ fx \doteq t \ \&\ C \ \to\ fx \doteq s \ \&\ s \doteq t \ \&\ C.$

A clause is called **normal** if it is unfolded and no normalization rule applies to it.

Proposition 3.7 *Let C be an unfolded feature clause. Then:*

1. *if D is obtained from C by a normalization rule, then D is an unfolded feature clause that is equivalent to C*
2. *there is no infinite chain of normalization steps issuing from C.*

Proof: The verification of the first claim is straightforward. To show the second claim, suppose there is an infinite sequence $C_1, C_2, \cdots$ of unfolded feature clauses such that, for every $i \geq 1$, C_{i+1} is obtained from C_i by a normalization rule. First note that every variable occurring in some C_i must also occur in C_1, that is, normalization steps don't introduce new variables. A variable x is called isolated in a clause C if C contains an equation $x \doteq s$ and x occurs exactly once in C. Now observe that no normalization rule decreases the number of isolated variables, and that the second normalization rule increases this number. Hence we can assume without loss of generality that the infinite sequence doesn't employ the second normalization rule. However, it is easy to see that the remaining normalization rules cannot support an infinite sequence. ■

Proposition 3.8 *For every feature clause one can compute in quadratic time a $\mathcal{V}C$-equivalent normal feature clause.*

Proof: Let C be a clause. We have seen that we can compute in linear time an unfolded clause D that is $\mathcal{V}C$-equivalent to C. By the previous proposition we know that we can compute a normal feature clause E that is equivalent to D using the normalization rules. The normalization of D to E can be done in quadratic time by employing the normalization rules together with an efficient union-find method [2] for maintaining equivalence classes of variables and constants. ■

A **clash** is an equation that has either the form $fa = s$ or the form $a \doteq b$, where a and b are different constants. A feature clause is **clash-free** if it contains no clash.

Proposition 3.9 *If a feature clause has a solution in some feature algebra, then it is clash-free. Furthermore, a feature clause is solved if and only if it is normal and clash-free.*

Now Theorem 3.2 follows easily from Propositions 3.8 and 3.9 and Theorem 3.6.

References

[1] G. Abrett and M. H. Burstein. The KREME knowledge editing environment. *International Journal of Man-Machine Studies*, 27(2):103–126, 1987.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.

[3] H. Aït-Kaci and R. Nasr. LOGIN: a logic programming language with built-in inheritance. *The Journal of Logic Programming*, 3:185–215, 1986.

[4] Y. Arens, L. Miller, S. C. Shapiro, and N. K. Sondheimer. Automatic construction of user-interface displays. In *Proceedings of the 7th National Conference of the American Association for Artificial Intelligence*, pages 808–813, Saint Paul, Minn., Aug. 1988.

[5] H. W. Beck, S. K. Gala, and S. B. Navathe. Classification as a query processing technique in the CANDIDE semantic data model. In *Proceedings of the International Data Engineering Conference, IEEE*, pages 572–581, Los Angeles, Cal., Feb. 1989.

[6] R. J. Brachman, A. Borgida, D. L. McGuinness, and L. A. Resnick. The CLASSIC knowledge representation system, or, KL-ONE: the next generation. In *Preprints of the Workshop on Formal Aspects of Semantic Networks*, Two Harbors, Cal., Feb. 1989.

[7] R. J. Brachman and H. J. Levesque. The tractability of subsumption in frame-based description languages. In *Proceedings of the 4th National Conference of the American Association for Artificial Intelligence*, pages 34–37, Austin, Tex., Aug. 1984.

[8] 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, Los Angeles, Cal., Aug. 1985.

[9] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, Apr. 1985.

[10] T. W. Finin and D. Silverman. Interactive classification as a knowledge acquisition tool. In L. Kerschberg, editor, *Expert Database Systems—Proceedings From the 1st International Workshop*, pages 79–90, Benjamin/Cummings, Menlo Park, Cal., 1986.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, Cal., 1979.

[12] M. Höhfeld and G. Smolka. *Definite Relations over Constraint Languages*. LILOG Report 53, IBM Deutschland, Stuttgart, West Germany, Oct. 1988.

[13] J. Jaffar and J. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, ACM, Munich, West Germany, Jan. 1987.

[14] M. Johnson. *Attribute-Value Logic and the Theory of Grammar*. *CSLI Lecture Notes 16*, Center for the Study of Language and Information, Stanford University, 1987.

[15] T. S. Kaczmarek, R. Bates, and G. Robins. Recent developments in NIKL. In *Proceedings of the 5th National Conference of the American Association for Artificial Intelligence*, pages 978–987, Philadelphia, Pa., Aug. 1986.

[16] P. C. Kanellakis and J. C. Mitchell. Polymorphic unification and ML typing. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages*, pages 5–15, Jan. 1989.

[17] R. Kaplan and J. Bresnan. Lexical-Functional Grammar: a formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–381, MIT Press, Cambridge, Mass., 1982.

[18] R. T. Kasper and W. C. Rounds. A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the ACL, Columbia University*, pages 257–265, New York, N.Y., 1986.

[19] M. Kay. Functional grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, Berkeley Linguistics Society, Berkeley, Cal., 1979.

[20] M. Kay. Parsing in functional unification grammars. In D. Dowty and L. Karttunen, editors, *Natural Language Parsing*, Cambridge University Press, Cambridge, England, 1985.

[21] M. Kay. *Unification Grammar*. Technical Report, Xerox PARC, Palo Alto, Cal., 1983.

[22] R. A. Kowalski. Algorithm = Logic + Control. *Communications of the ACM*, 22(7):424–436, 1979.

[23] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[24] R. MacGregor and R. Bates. *The Loom Knowledge Representation Language*. Technical Report ISI/RS-87-188, University of Southern California, Information Science Institute, Marina del Rey, Cal., 1987.

[25] K. Mukai. *Anadic Tuples in Prolog.* Technical Report TR-239, ICOT, Tokyo, Japan, 1987.

[26] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, Apr. 1988.

[27] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* PhD thesis, Universität des Saarlandes, Saarbrücken, West Germany, June 1989.

[28] B. Nebel. *Terminological Reasoning is Inherently Intractable.* IWBS Report, IWBS, IBM Deutschland, Stuttgart, West Germany, Aug. 1989. In preparation.

[29] B. Nebel and K. von Luck. Hybrid reasoning in BACK. In Z. W. Ras and L. Saitta, editors, *Methodologies for Intelligent Systems*, pages 260–269, North-Holland, Amsterdam, Holland, 1988.

[30] R. Neches, W. R. Swartout, and J. D. Moore. Explainable (and maintainable) expert systems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 382–389, Los Angeles, Cal., Aug. 1985.

[31] B. Owsnicki-Klewe. Configuration as a consistency maintenance task. In W. Hoeppner, editor, *GWAI-88. 12th German Workshop on Artificial Intelligence*, pages 77–87, Springer-Verlag, Berlin, West Germany, 1988.

[32] P. F. Patel-Schneider. A four-valued semantics for terminological logics. *Artificial Intelligence*, 38(3):319–351, Apr. 1989.

[33] 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.

[34] P. F. Patel-Schneider. Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39(2):263–272, June 1989.

[35] P. F. Patel-Schneider, R. J. Brachman, and H. J. Levesque. ARGON: knowledge representation meets information retrieval. In *Proceedings of the 1st Conference on Artificial Intelligence Applications*, pages 280–286, Denver, Col., 1984.

[36] F. Pereira. Grammars and logics of partial information. In *Proceedings of the 4th International Conference on Logic Programming*, pages 989–1013, MIT Press, Cambridge, Mass., 1987.

[37] F. Pereira and D. Warren. Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[38] C. Pollard and I. Sag. *An Information-Based Syntax and Semantics. CSLI Lecture Notes 13*, Center for the Study of Language and Information, Stanford University, 1987.

[39] W. Rounds and R. Kasper. A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 1st IEEE Symposium on Logic in Computer Science*, pages 38–43, Boston, Mass., 1986.

[40] W. C. Rounds and A. Manaster-Ramer. A logical version of functional grammar. In *Proceedings of the 25th Annual Meeting of the ACL, Stanford University*, pages 89–96, Stanford, Cal., 1987.

[41] K. Schild. *Undecidability of $\mathcal{U}$.* KIT Report 67, Department of Computer Science, Technische Universität Berlin, Berlin, West Germany, Oct. 1988.

[42] M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, Toronto, Ont., May 1989.

[43] M. Schmidt-Schauß and G. Smolka. *Attributive Concept Descriptions with Unions and Complements.* SEKI Report SR-88-21, Department of Computer Science, Universität Kaiserslautern, Kaiserslautern, West Germany, Dec. 1988.

[44] S. M. Shieber. The design of a computer language for linguistic information. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 362–366, Stanford, Cal., 1984.

[45] S. M. Shieber. *An Introduction to Unification-Based Approaches to Grammar. CSLI Lecture Notes 4*, Center for the Study of Language and Information, Stanford University, 1986.

[46] S. M. Shieber. *Parsing and Type Inference for Natural and Computer Languages.* Technical Note 460, SRI International, Artificial Intelligence Center, Menlo Park, Cal., March 1989.

[47] S. M. Shieber, H. Uszkoreit, F. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In J. Bresnan, editor, *Research on Interactive Acquisition and Use of Knowledge*, SRI International, Artificial Intelligence Center, Menlo Park, Cal., 1983.

[48] G. Smolka. *A Feature Logic with Subsorts.* LILOG Report 33, IBM Deutschland, Stuttgart, May 1988.

[49] N. K. Sondheimer and B. Nebel. A logical-form and knowledge-base design for natural language generation. In *Proceedings of the 5th National Conference of the American Association for Artificial Intelligence*, pages 612–618, Philadelphia, Pa., Aug. 1986.

[50] F. N. Tou, M. D. Williams, R. E. Fikes, A. Henderson, and T. Malone. RABBIT: an intelligent database assistant. In *Proceedings of the 2nd National Conference of the American Association for Artificial Intelligence*, pages 314–318, Pittsburgh, Pa., Aug. 1982.

[51] M. B. Vilain. The restricted language architecture of a hybrid representation system. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 547–551, Los Angeles, Cal., Aug. 1985.

[52] W. A. Woods. What's important about knowledge representation. *IEEE Computer*, 16(10):22–29, Oct. 1983.