A Feature Logic with Subsorts*

Gert Smolka

German Research Center for Artificial Intelligence and
Universität des Saarlandes
Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany
smolka@dfki.uni-sb.de

Abstract

This paper presents a set description logic with subsorts, feature selection (the inverse of unary function application), agreement, intersection, union and complement. We define a model theoretic open world semantics and show that sorted feature structures constitute a canonical model, that is, without loss of generality subsumption and consistency of set descriptions can be considered with respect to feature structures only. We show that deciding consistency of set descriptions is an NP-complete problem.

To appear in:
J. Wedekind and C. Rohrer (eds.), Unification in Grammar.
The MIT Press, 1992

*This text is a minor revision of LILOG Report 33, May 1988, IBM Deutschland, IWBS, Postfach 800880, 7000 Stuttgart 80, Germany. The research reported here has been done while the author was with IBM Deutschland. The author's article [23] is a more recent work on feature logics.

1 Introduction

This paper presents a set description logic that generalizes and integrates formalisms that have been developed for knowledge representation [1, 2] and computational linguistics [13, 21, 11, 9]. The logic comes with an open world model theoretic semantics, where admissible worlds can be required to satisfy a classification scheme postulated by means of a sort lattice. The logic supports the typically partial description of objects using sorts and features as primitives. It is not intended as a stand-alone formalism but as descriptive component of more expressive formalisms like unification grammars [22], knowledge representation systems or logic programming languages.

The logic is based on the notion of a *signature*, which specifies which interpretations are admissible. A signature postulates a set of feature symbols and a lattice of sort symbols, where $\bot$ is the least and $\top$ is the greatest sort symbol. Figure 1 shows an example of a sort lattice taken form [2].

An *interpretation* of a signature assigns to every sort symbol a set, where the denotation of $\top$ is called the *universe* of the interpretation, and to every feature symbol a partial unary function from the universe to the universe. The denotations of the sort symbols must satisfy the following conditions:

- $\bot$ denotes the empty set
- if C is the greatest common subsort of two sorts A and B, then C denotes the intersection of the denotations of A and B.

This definition implies that the denotation of A is a subset of the denotation of B if A is a subsort of B. Furthermore, if the greatest common subsort of A and B is $\bot$, then A and B must denote disjoint sets.

Given a signature specifying a set $\mathbf{S}$ of sort symbols and a set $\mathbf{F}$ of feature symbols, one possible interpretation can be obtained by taking the least solution of the domain equation

$$\mathbf{U} = (\mathbf{S} \perp \{\bot\}) \times (\mathbf{F} \rightarrow \mathbf{U})$$

as the universe, where $\mathbf{F} \rightarrow \mathbf{U}$ stands for the set of all finite partial functions from $\mathbf{F}$ to $\mathbf{U}$. The elements of this universe are finite records labeled with a sort symbol different from $\bot$ whose fields are designated with feature symbols. Figure 2 shows an example of such a record. An interpretation with this universe interprets a sort symbol A as the set of all records labeled with a

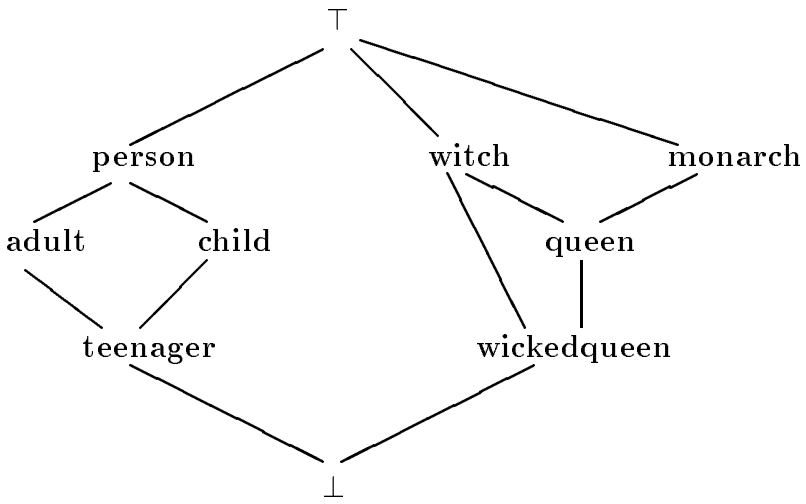

Figure 1: An example of a sort lattice.

$$\left[\begin{array}{l} \textbf{clause} \\ \textit{agent}: \left[\begin{array}{l} \textbf{group} \\ \textit{primaryclass}: \textbf{nominal} \\ \textit{animacy}: \textbf{animate} \end{array}\right] \\ \textit{process}: \left[\begin{array}{l} \textbf{group} \\ \textit{primaryclass}: \textbf{verbal} \\ \textit{agency}: \textbf{effective} \end{array}\right] \end{array}\right]$$

Figure 2: A record written in matrix notation. Sort symbols are printed bold and feature symbols are printed slanted.

subsort of A, and a feature symbol f as the partial function yielding the value of the field f if it is present.

Given a signature, the logic offers several languages for describing subsets of the universe of an interpretation. The most intuitive of these languages is applicative and consists of expressions called *feature terms*, which can be viewed as a more general form of sorts. The sort symbols of the signature are taken as primitive feature terms and operators providing for set intersection, union and complement with respect to the universe are available. For instance, the feature term

$$(A \sqcap \neg B) \sqcup (B \sqcap \neg A)$$

denotes the set of all elements that are in the union of the sorts A and B but not in the intersection of A and B.

There are two further, more specific term forming operators that rely on the presence of features.

The *selection operator* takes a feature symbol f and a feature term s and yields a feature term $f\!:\!s$ denoting the set of all elements of the universe for which the feature f is defined and yields an element of the set denoted by s. Feature selection is the inverse of unary function application. For instance, the term $\neg(f\!:\top)$ denotes the set of all elements of the universe for which the feature f is not defined. If we interpret this term over the universe of finite records defined above, it denotes all records that don't have a field f at the top level. The feature term

$$\mathbf{car} \sqcap \mathit{fuel}\!:\mathbf{gasoline}$$

denotes the set of all elements of the sort **car**, for which the feature *fuel* is defined and yields an element of the sort **gasoline**. In the domain of finite records, this term denotes the set of all records labeled with a subsort of **car** that possess a field *fuel* having as its value a record labeled with a subsort of **gasoline**. One such record is

$$\begin{bmatrix} \mathbf{car} \\ \mathit{fuel}\!:\mathbf{gasoline} \\ \mathit{speed}\!:\mathbf{120} \end{bmatrix}.$$

The last term forming operator is the *agreement operator* $p \downarrow q$ taking two paths as arguments. A *path* is a finite, possible empty sequence $f_1 \cdots f_n$ of feature symbols denoting the composition of the partial functions denoted by $f_1, \ldots, f_n$, where f_1 is applied first. A feature term $p \downarrow q$ denotes the set

of all elements of the universe for which the denotations of p and q are both defined and yield the same element as result. For instance, the term

$$\mathbf{car} \sqcap \mathit{speed} \downarrow \mathit{age}$$

denotes the set of all elements of the sort **car** for which the features *speed* and *age* are both defined and agree. (Cars in this set are the slower the newer they are.) Linguists will prefer the term

$$\mathbf{sentence} \sqcap (\mathit{subj}\ \mathit{agree}) \downarrow (\mathit{pred}\ \mathit{agree})$$

which may be thought of as denoting all sentences possessing a subject and a predicate with agreeing *agree* features.

A feature term is called *consistent* if there exists a least one admissible interpretation in which it denotes a nonempty term. For instance, the feature term

$$f{:}A \sqcap f{:}B$$

is inconsistent if the greatest common subsort of A and B is $\bot$. However, if the greatest common subsort of A and B is a sort symbol C different from $\bot$, then the term is consistent since, for instance, it contains $[\top\ f{:}C]$ in the record interpretation.

We say that a feature term s is *subsumed by* a feature term t if s denotes a subset of t in every admissible interpretation. Furthermore, we say that two terms are *equivalent* if they denote the same set in every admissible interpretation. Since the logic has negation, consistency and subsumption are closely related properties: s is inconsistent if and only if s is subsumed by $\bot$, and s is subsumed by t if and only if $s \sqcap \neg t$ is inconsistent.

We will show that deciding the consistency of feature terms is an NP-complete problem. The NP-hardness is shown using Kasper's [11] reduction of the satisfiability problem of propositional formulas in conjunctive normal form to the consistency problem of feature terms. This reduction shows in particular that the consistency problem remains NP-hard even if only feature terms without complements and agreements are considered.

Several variants of feature descriptions are being used in computational linguistics (see [22, 19, 20] for introductory expositions). In unification grammars [10, 14], unification of feature descriptions is the basic operational mechanism for parsing and generating natural language. A unification method for feature descriptions consists of a normal form that exhibits inconsistency and an algorithm that, given two normal feature descriptions s and t, computes a normal feature description equivalent to $s \sqcap t$.

Rounds and Kasper [21] were the first to come up with a logical formalization of feature descriptions (without complements and with constants rather than sorts). Their logic has a fixed interpretation, where feature descriptions denote sets of feature structures. Feature structures are consistent, union-free feature descriptions that can be viewed as finite deterministic automata. This paper generalizes Rounds and Kasper's logic in two respects: first, we add complements and nonsingleton sorts; and second, and maybe more important, we show that our open world semantics yields the same notion of subsumption and consistency as a closed world semantics using feature structures as fixed interpretation.

Moshier and Rounds [16] generalize the formalism of Rounds and Kasper [21] by adding nonclassical negation. They show that the consistency problem of the extended logic is PSPACE-complete. In contrast, classical negation, which underlies our logic, does not render the consistency problem any harder (it is already NP-complete if just singletons, selection and union are available).

Johnson's [9] attribute value logic is close to the unification grammar formalism LFG [10]. It employs relational feature descriptions with classical negation, where agreement is expressed with variables. Like Rounds and Kasper's logic, Johnson's logic is interpreted over feature structures only. In contrast to Rounds and Kasper, Johnson admits cyclic feature structures (as do we). Johnson shows that the consistency problem of his logic is NP-complete. In this paper, we will generalize feature terms to accommodate variables and thus obtain feature descriptions that are almost equivalent to Johnson's (Johnson's descriptions are more general in that they allow for variables at the left-hand side of selections). We will show that feature terms with variables describe exactly the same sets as variable-free feature terms, thus bridging the gap between Johnson's and Rounds and Kasper's formalisms. However, feature terms with variables allow for exponentially more succinct descriptions, a fact that is important for efficient unification algorithms [12, 6]. We will also introduce quantification for feature terms with variables and present a quantifier elimination algorithm that computes for every feature term an equivalent quantifier-free feature term, thus showing that even with quantification no more sets can be described and consistency and subsumption remain decidable.

Our feature logic generalizes Aït-Kaci's formalism [1, 2], from which it inherits the use of subsorts. Aït-Kaci's formalism was developed independently of the linguistically motivated approaches with knowledge representation as application in mind. From today's perspective, Aït-Kaci's formalism suffers

from the fact that it is nonlogical, that is, that it makes no clear distinction between descriptions and what they denote. There are only feature descriptions in normal form (called ψ-terms) and subsumption is defined purely syntactical. Nevertheless, Aït-Kaci was the first to give a mathematically rigorous formalization of what feature unification is supposed to do. Aït-Kaci [1, 2] also outlines a model theoretic semantics for his formalism that is equivalent to the semantics presented in this paper, but he doesn't exploit this idea any further (for instance, for the definition of subsumption).

In [24] we give an initial algebra semantics (closed world) for feature descriptions drawn over inheritance hierarchies using order-sorted equational logic [25]. This approach accommodates data types whose elements are described by features as well as data types whose elements are described by constructors and shows the duality of the two approaches under the given initial algebra semantics. Furthermore, we present a unification algorithm that combines order-sorted unification [27, 28] with ψ-term unification [1, 3].

The paper is organized as follows. Section 2 formalizes signatures, interpretations and feature terms and shows that every feature term can be rewritten as a union of simple feature terms. Section 3 presents a relational language of set descriptions and shows that every simple feature term can be represented as a normal set description. The translation is accomplished by a system of simplification rules, which provides a consistency checking and unification algorithm. Section 4 shows that feature structures constitute a canonical interpretation and that deciding consistency of feature terms is an NP-complete problem. Section 5 generalizes feature terms to include variables and quantification, which provide for an exponentially more succinct syntax. Finally, Section 6 discusses the results, possible applications and related work. It might be a good idea to skim this discussion before getting lost in the technical sections of the paper.

Acknowledgement. Günther Görz and Hans Uszkoreit were patient enough to give me some idea of what is going on in computational linguistics. Hans Uszkoreit's STUF [26], which is used as the central representation formalism in the LILOG project at IBM, provided me with the challenge to capture some of it in logic. From Mark Johnson's thesis I learned that complements and negations are easy. Finally, Christoph Beierle and Markus Höhfeld helped me with frequent discussions to get things right.

2 Feature Terms

In this section we define the basic notions of feature logic: signatures, interpretations and feature terms. Feature terms are a functional language for feature logic. In later sections we will introduce a relational language and extend the functional language to include variables and quantification.

Signatures serve as the interface between syntax (formal languages) and semantics (interpretations). In our logic, a signature fixes the available symbols (sorts, singletons and features) together with a subsort ordering.

Formally, a *signature* is a tuple $\Sigma = (\mathbf{S}, \mathbf{C}, \leq, \mathbf{F})$ specifying

- a set $\mathbf{S}$ of *sort symbols* containing $\bot$ and $\top$
- a subset $\mathbf{C}$ of $\mathbf{S}$ whose elements are called *singleton symbols*
- a decidable partial order $\leq$ on $\mathbf{S}$ such that
  - $\bot$ is the least and $\top$ is the greatest element
  - every two sort symbols A and B of $\mathbf{S}$ have a greatest common lower bound, which is called their *greatest common subsort* and is denoted by $\mathbf{gcs}(A, B)$
  - every singleton symbol A is minimal, that is, if $B \leq A$, then B is either $\bot$ or A
- a set $\mathbf{F}$ of *feature symbols* such that $\mathbf{S}$ and $\mathbf{F}$ are disjoint.

An *interpretation* $\mathcal{A}$ of a signature Σ (also called a Σ-*algebra*) consists of denotations $A^{\mathcal{A}}$ and $f^{\mathcal{A}}$ for the sort and feature symbols of Σ such that:

- $\top^{\mathcal{A}}$ is a set called the *universe of* $\mathcal{A}$
- $\bot^{\mathcal{A}}$ is the empty set
- if A is a sort symbol of Σ, then $A^{\mathcal{A}}$ is a subset of $\top^{\mathcal{A}}$
- if A is a singleton symbol of Σ, then $A^{\mathcal{A}}$ is a set consisting of exactly one element
- if A and B are sort symbols of Σ having C as their greatest common subsort, then $C^{\mathcal{A}} = A^{\mathcal{A}} \cap B^{\mathcal{A}}$

- if f is a feature symbol of Σ, then $f^{\mathcal{A}}$ is a function $\mathcal{D}_f^{\mathcal{A}} \to \top^{\mathcal{A}}$, where $\mathcal{D}_f^{\mathcal{A}}$ (called the *domain of f in $\mathcal{A}$*) is a subset of $\top^{\mathcal{A}}$
- if f is a feature symbol of Σ and A is a singleton symbol of Σ, then $\mathcal{D}_f^{\mathcal{A}}$ and $A^{\mathcal{A}}$ are disjoint.

This definition ensures that different singleton symbols denote disjoint singletons and that no feature is defined on a singleton. Thus our singleton symbols can take exactly the rôle of the constant symbols employed in the feature logics of Rounds and Kasper [21] and Johnson [9].

Note that the subsort ordering of a signature must be realized by an interpretation in a strong sense: if C is the greatest common subsort of A and B, then the denotation of C must be the intersection (and not merely a subset) of the denotations of A and B. With that we are able to postulate that two sorts are disjoint by making $\bot$ their only common subsort.

In the following it won't be necessary to refer to more than one signature at once. Thus we will ease our notation by always referring to some fixed signature $\Sigma = (\mathbf{S}, \mathbf{C}, \leq, \mathbf{F})$. Furthermore, the letters A and B will always denote sort symbols of $\mathbf{S}$ and the letters f, g and h will always denote feature symbols of $\mathbf{F}$.

A *path* is a finite sequence of feature symbols. The *empty path* is denoted by ϵ. In an interpretation $\mathcal{A}$, a path $f_1 \cdots f_n$ denotes the partial function obtained as the composition of $f_1^{\mathcal{A}}, \ldots, f_n^{\mathcal{A}}$, where $f_1^{\mathcal{A}}$ is applied first:

- $\epsilon^{\mathcal{A}}$ is the identity function on $\top^{\mathcal{A}}$
- $\mathcal{D}_{fq}^{\mathcal{A}} := \{a \in \mathcal{D}_f^{\mathcal{A}} \mid f^{\mathcal{A}}(a) \in \mathcal{D}_q^{\mathcal{A}}\}$
- $(fq)^{\mathcal{A}}(a) := q^{\mathcal{A}}(f^{\mathcal{A}}(a))$.

The letters p and q will always denote paths.

Feature terms are defined by the following context-free production rule:

$$
\begin{array}{lcll}
s, t, u, v & \bot\!\!\to & A & \textit{sort} \\
& | & f\!:\!s & \textit{selection} \\
& | & p \downarrow q & \textit{agreement} \\
& | & p \uparrow q & \textit{disagreement} \\
& | & s \sqcap t & \textit{intersection} \\
& | & s \sqcup t & \textit{union} \\
& | & \neg s & \textit{complement.}
\end{array}
$$

The *denotation* of a feature term in an interpretation $\mathcal{A}$ is a subset of $\top^{\mathcal{A}}$ defined inductively by the following equations:

- $[\![A]\!]^{\mathcal{A}} = A^{\mathcal{A}}$
- $[\![f{:}\, s]\!]^{\mathcal{A}} = \{a \in \mathcal{D}_f^{\mathcal{A}} \mid f^{\mathcal{A}}(a) \in [\![s]\!]^{\mathcal{A}}\} = (f^{\mathcal{A}})^{-1}([\![s]\!]^{\mathcal{A}})$
- $[\![p \downarrow q]\!]^{\mathcal{A}} = \{a \in \mathcal{D}_p^{\mathcal{A}} \cap \mathcal{D}_q^{\mathcal{A}} \mid p^{\mathcal{A}}(a) = q^{\mathcal{A}}(a)\}$
- $[\![p \uparrow q]\!]^{\mathcal{A}} = \{a \in \mathcal{D}_p^{\mathcal{A}} \cap \mathcal{D}_q^{\mathcal{A}} \mid p^{\mathcal{A}}(a) \neq q^{\mathcal{A}}(a)\}$
- $[\![s \sqcap t]\!]^{\mathcal{A}} = [\![s]\!]^{\mathcal{A}} \cap [\![t]\!]^{\mathcal{A}}$
- $[\![s \sqcup t]\!]^{\mathcal{A}} = [\![s]\!]^{\mathcal{A}} \cup [\![t]\!]^{\mathcal{A}}$
- $[\![\neg s]\!]^{\mathcal{A}} = \top^{\mathcal{A}} \perp [\![s]\!]^{\mathcal{A}}$.

To ease our notation, we will omit parentheses whenever the following rules allow for disambiguation:

- $\sqcap$ and $\sqcup$ are right-associative
- the term forming operators bind according to the order $f{:}\, s$, $\neg s$, $s \sqcap t$, $s \sqcup t$, where selection binds strongest and union binds weakest.

For instance,

$$\neg f{:}\, p \downarrow q \sqcup s \sqcap \neg t \sqcap u \sqcup v$$

disambiguates to

$$(\neg(f{:}\,(p \downarrow q))) \sqcup ((s \sqcap ((\neg t) \sqcap u)) \sqcup v).$$

Furthermore, we will use the following abbreviations:

- $s \perp t := s \sqcap \neg t$ *(set difference)*
- $p{:}\, s := \begin{cases} s & \text{if } p = \epsilon \\ f{:}\,(q{:}\, s) & \text{if } p = fq. \end{cases}$

Disagreements and unions are actually redundant forms since

$$\begin{aligned} [\![p \uparrow q]\!]^{\mathcal{A}} &= [\![p{:}\,\top \sqcap q{:}\,\top \sqcap \neg\, p \downarrow q]\!]^{\mathcal{A}} \\ [\![s \sqcup t]\!]^{\mathcal{A}} &= [\![\neg(\neg s \sqcap \neg\, t)]\!]^{\mathcal{A}} \end{aligned}$$

in every interpretation $\mathcal{A}$. They have been introduced as separate syntactic forms since they ease the definition of transformations to be introduced below.

Given an interpretation $\mathcal{A}$, the system of all subsets of the universe of $\mathcal{A}$ that can be obtained as the denotation of a feature term

$$\mathbf{FTS}(\mathcal{A}) := \{[\![s]\!]^{\mathcal{A}} \mid s \text{ is a feature term}\}$$

is a boolean set lattice since it is closed under intersection, union and complement and contains $\bot^{\mathcal{A}} = \emptyset$ and the universe $\top^{\mathcal{A}}$.

We are now ready to define the major properties of feature terms. We say that

- a feature term s is *consistent* if there exists an interpretation $\mathcal{A}$ such that $[\![s]\!]^{\mathcal{A}} \neq \emptyset$
- a feature term s is *subsumed* by a feature term t if $[\![s]\!]^{\mathcal{A}} \subseteq [\![t]\!]^{\mathcal{A}}$ for every interpretation $\mathcal{A}$
- a feature term s is *equivalent* to a feature term t if $[\![s]\!]^{\mathcal{A}} = [\![t]\!]^{\mathcal{A}}$ for every interpretation $\mathcal{A}$.

Proposition 2.1 (Tautologies) *Let s and t be feature terms and $u \doteq v$ be an instance of one of the tautologies in Figure 3. If t is obtainable from s by replacing a subterm u with v, then s and t are equivalent.*

Proposition 2.2 (Reductions) *Let s and t be feature terms. Then:*

- *s is subsumed by t* $\iff$ *$s \perp t$ is inconsistent*
  $\iff$ *s and $s \sqcap t$ are equivalent*
- *s is equivalent to t* $\iff$ *$s \perp t \sqcup t \perp s$ is inconsistent*
  $\iff$ *s is subsumed by t and t is subsumed by s*
- *s is inconsistent* $\iff$ *s is subsumed by $\bot$*
  $\iff$ *s is equivalent to $\bot$.*

Commutativity, Associativity and Distributivity

$s \sqcap t \doteq t \sqcap s$ $\qquad$ $s \sqcup t \doteq t \sqcup s$

$s \sqcap (t \sqcap u) \doteq (s \sqcap t) \sqcap u$ $\qquad$ $s \sqcup (t \sqcup u) \doteq (s \sqcup t) \sqcup u$

$s \sqcap (t \sqcup u) \doteq (s \sqcap t) \sqcup (s \sqcap u)$ $\qquad$ $s \sqcup (t \sqcap u) \doteq (s \sqcup t) \sqcap (s \sqcup u)$

Idempotence and Absorption

$s \sqcap s \doteq s$ $\qquad$ $s \sqcup s \doteq s$

$(s \sqcap t) \sqcup s \doteq s$ $\qquad$ $(s \sqcup t) \sqcap s \doteq s$

Complements

$s \sqcap \neg s \doteq \bot$ $\qquad$ $s \sqcup \neg s \doteq \top$

$\neg(s \sqcap t) \doteq \neg s \sqcup \neg t$ $\qquad$ $\neg(s \sqcup t) \doteq \neg s \sqcap \neg t$

$\neg\neg s \doteq s$

Bottom and Top

$s \sqcap \bot \doteq \bot$ $\qquad$ $s \sqcup \top \doteq \top$

$s \sqcap \top \doteq s$ $\qquad$ $s \sqcup \bot \doteq s$

$\neg\bot \doteq \top$ $\qquad$ $\neg\top \doteq \bot$

$f\!:\bot \doteq \bot$

Selections and Agreements

$f\!:(s \sqcap t) \doteq (f\!:s) \sqcap (f\!:t)$ $\qquad$ $f\!:(s \sqcup t) \doteq (f\!:s) \sqcup (f\!:t)$

$\neg f\!:s \doteq (\neg f\!:\top) \sqcup (f\!:\neg s)$

$\neg p \downarrow q \doteq (\neg p\!:\top) \sqcup (\neg q\!:\top) \sqcup p \uparrow q$

$\neg p \uparrow q \doteq (\neg p\!:\top) \sqcup (\neg q\!:\top) \sqcup p \downarrow q$

Sorts

$A \sqcap B \doteq \mathbf{gcs}(A, B)$ $\qquad$ $A \sqcup B \doteq B$ if $A \leq B$

Figure 3: Some Feature Term Tautologies.

The Reduction Proposition says that a decision algorithm for one of the three properties subsumption, equivalence and consistency can be used for deciding any of these properties. Since the reductions are linear time, all three decision problems have the same computational complexity.

In the following sections we will concentrate on the consistency problem, which we will show to be NP-complete. The first step shows that every feature term can be rewritten in polynomial time to an equivalent feature term containing only simple complements.

A complement is called *simple* if it has either the form $\neg f{:}\top$ or the form $\neg A$, where A is a sort symbol different from $\bot$ and $\top$.

Proposition 2.3 (Simple complements) *For every feature term one can compute in polynomial time an equivalent feature term containing only simple complements by rewriting with the following tautologies in top-down order:*

1. $\neg\bot \doteq \top$
2. $\neg\top \doteq \bot$
3. $\neg f{:}s \doteq \neg f{:}\top \sqcup f{:}\neg s$
4. $\neg p \downarrow q \doteq \neg p{:}\top \sqcup \neg q{:}\top \sqcup p \uparrow q$
5. $\neg p \uparrow q \doteq \neg p{:}\top \sqcup \neg q{:}\top \sqcup p \downarrow q$
6. $\neg(s \sqcap t) \doteq \neg s \sqcup \neg t$
7. $\neg(s \sqcup t) \doteq \neg s \sqcap \neg t$
8. $\neg\neg s \doteq s$.

A feature term is called *simple* if it contains no union and every contained complement is simple.

Proposition 2.4 (Disjunctive Normal Form) *For every feature term s one can compute in exponential time finitely many simple feature terms $s_1, \ldots, s_n$ such that s and $s_1 \sqcup \cdots \sqcup s_n$ are equivalent by first rewriting all complements to simple complements and then propagating up all unions by rewriting with the following tautologies:*

1. $f{:}(s \sqcup t) \doteq (f{:}s) \sqcup (f{:}t)$
2. $s \sqcap (t \sqcup u) \doteq (s \sqcap t) \sqcup (s \sqcap u)$
3. $(s \sqcup t) \sqcap u \doteq (s \sqcap u) \sqcup (t \sqcap u)$.

The terms $s_1, \ldots, s_n$ are called the disjuncts *of s. The disjuncts of a term can also be obtained by first rewriting all complements to simple complements and then replacing every union nondeterministically by one of its arguments.*

In the following sections we will prove that the consistency of simple feature terms is decidable in polynomial time. By the Reduction and the Disjunctive Normal Form Proposition of this section we already know that this result implies that deciding consistency of general feature terms is in NP, and that deciding subsumption of general feature terms is in co-NP.

3 Feature Clauses and Simplification Rules

We won't provide a deduction calculus for proving feature term equivalences since this turns out to be a very tedious enterprise. The difficulties are caused by the presence of agreements. The interested reader may consult [21], where a complete deduction calculus for feature terms without complements and sorts is given.

The difficulties with the deduction calculus disappear if we use a relational rather than a functional language. A relational language has variables that range over the universe of the interpretation, primitive constraints, and conjunction, disjunction and negation to form complex constraints. As primitive constraints one can employ, for instance, the forms

- $x{:}A$ "x is in sort A"
- $f(x) \doteq y$ "feature f of x is y"
- $x \doteq y$ "x equals y".

A similar language, without sorts, is used in Johnson's [9] attribute value logic, which comes with a complete and sound deduction calculus.

Here we will be content to devise a simple relational language that is just powerful enough to describe sets that can be obtained as the denotation of simple feature terms. Using this relational language, we can conveniently specify a polynomial-time algorithm for deciding the consistency of simple feature terms.

From now on we assume that an infinite alphabet of *variables* is given. Of course, variables are assumed to be distinct from sort and feature symbols. The letters x, y and z will always denote variables.

Constraints are defined by the following context-free production rule:

$c \ \bot\!\!\!\rightarrow \quad x{:}s, \qquad$ where s is simple*containment*

	$xp \doteq y$	*path equation*
	$x \doteq y$	*variable equation*
	$f(x) \doteq y$	*feature equation*
	$x \not\doteq y$	*disequation.*

To ease our notation, we will write $\neg f(x)$ (read "f is not defined on x") for a containment $x\colon(\neg f\colon\top)$.

Let $\mathcal{A}$ be an interpretation. An *$\mathcal{A}$-assignment* α is a function that maps every variable to an element of $\top^{\mathcal{A}}$. The *validity* of a constraint in an interpretation $\mathcal{A}$ under an $\mathcal{A}$-assignment α is defined as follows:

- $\mathcal{A}, \alpha \models x\colon s \quad :\Longleftrightarrow \quad \alpha(x) \in [\![s]\!]^{\mathcal{A}}$
- $\mathcal{A}, \alpha \models xp \doteq y \quad :\Longleftrightarrow \quad \alpha(x) \in \mathcal{D}_p^{\mathcal{A}}$ and $p^{\mathcal{A}}(\alpha(x)) = \alpha(y)$
- $\mathcal{A}, \alpha \models x \doteq y \quad :\Longleftrightarrow \quad \alpha(x) = \alpha(y)$
- $\mathcal{A}, \alpha \models f(x) \doteq y \quad :\Longleftrightarrow \quad \alpha(x) \in \mathcal{D}_f^{\mathcal{A}}$ and $f^{\mathcal{A}}(\alpha(x)) = \alpha(y)$
- $\mathcal{A}, \alpha \models x \not\doteq y \quad :\Longleftrightarrow \quad \alpha(x) \neq \alpha(y)$.

A *feature clause* is a finite, possibly empty set of constraints. A feature clause C is *valid* in an interpretation $\mathcal{A}$ under an $\mathcal{A}$-assignment α if $\mathcal{A}, \alpha \models c$ for every constraint $c \in C$.

A *set description* is a pair $x|C$ consisting of a variable x and a feature clause C. The *denotation* of a set description $x|C$ in an interpretation $\mathcal{A}$ is defined as

$$[\![x|C]\!]^{\mathcal{A}} := \{\alpha(x) \mid \alpha \text{ is an } \mathcal{A}\text{-assignment such that } \mathcal{A}, \alpha \models C\}.$$

A set description is called *consistent* if there exists at least one interpretation in which it denotes a nonempty set. Two set descriptions are called *equivalent* if if they denote the same set in every interpretation; furthermore, a feature term and a set descriptions are called *equivalent* if if they denote the same set in every interpretation.

Proposition 3.1 *Let s be a simple feature term and let x be a variable. Then $x|\{x\colon s\}$ is a set description such that $[\![s]\!]^{\mathcal{A}} = [\![x|\{x\colon s\}]\!]^{\mathcal{A}}$ for every interpretation $\mathcal{A}$.*

(1) $x{:}\,(f{:}\,s)\ \&\,C \rightarrow_{x_o} f(x) \doteq y\ \&\,y{:}\,s\ \&\,C$ if y is new

(2) $x{:}\,(p \downarrow q)\ \&\,C \rightarrow_{x_o} xp \doteq y\ \&\,xq \doteq y\ \&\,C$ if y is new

(3) $x{:}\,(p \uparrow q)\ \&\,C \rightarrow_{x_o} xp \doteq y\ \&\,xq \doteq z\ \&\,y \not\doteq z\ \&\,C$
if y and z are new and distinct

(4) $x{:}\,(s \sqcap t)\ \&\,C \rightarrow_{x_o} x{:}\,s\ \&\,x{:}\,t\ \&\,C$

(5) $xfp \doteq y\ \&\,C \rightarrow_{x_o} f(x) \doteq z\ \&\,zp \doteq y\ \&\,C$ if z is new

(6) $x\epsilon \doteq y\ \&\,C \rightarrow_{x_o} x \doteq y\ \&\,C$

Figure 4: The simplification rules for decomposing containments.

Next we will define a normal form for set descriptions that exhibits inconsistency. Then we will present simplification rules with which every set description can be transformed to normal form in polynomial time.

A constraint is called *normal* if it has one of the following forms:

- $x{:}\,A$ or $x{:}\,\neg A$, where A is a sort symbol different from $\bot$ and $\top$
- $\neg f(x)$ (abbreviation for $x{:}\,(\neg f{:}\,\top)$)
- $f(x) \doteq y$
- $x \not\doteq y$, where x and y are distinct variables.

A feature clause is called *normal* if either it has the form

$$\{x{:}\,\bot\}$$

or it is a set of normal constraints satisfying the following conditions:

1. if $x{:}\,A$ and $x{:}\,B$ are in C, then $A = B$
2. if $x{:}\,A$ and $x{:}\,\neg B$ are in C, then A and B are distinct and A is not a subsort of B
3. if $x{:}\,\neg A$ and $x{:}\,\neg B$ are in C, then $A = B$ or A is not a subsort of B
4. if $x{:}\,A$ and $y{:}\,A$ are in C and A is a singleton, then $x = y$
5. if $f(x) \doteq y$ and $f(x) \doteq z$ are in C, then $y = z$

Sorts

(7) $x{:}\,A \mathbin{\&} x{:}\,B \mathbin{\&} C \;\rightarrow_{x_o}\; x{:}\,\mathbf{gcs}(A,B) \mathbin{\&} C$

(8) $x{:}\,\top \mathbin{\&} C \;\rightarrow_{x_o}\; C$

(9) $x{:}\,\bot \mathbin{\&} C \;\rightarrow_{x_o}\; x_o{:}\,\bot$ if C is nonempty

(10) $x{:}\,\neg A \mathbin{\&} x{:}\,\neg B \mathbin{\&} C \;\rightarrow_{x_o}\; x{:}\,\neg B \mathbin{\&} C$ if $A \leq B$

(11) $x{:}\,A \mathbin{\&} x{:}\,\neg B \mathbin{\&} C \;\rightarrow_{x_o}\; x_o{:}\,\bot$ if $A \leq B$

(12) $x{:}\,A \mathbin{\&} f(x) \doteq y \mathbin{\&} C \;\rightarrow_{x_o}\; x_o{:}\,\bot$ if A is a singleton

(13) $x{:}\,A \mathbin{\&} y{:}\,A \mathbin{\&} C \;\rightarrow_{x_o}\; y{:}\,A \mathbin{\&} [x/y]C$ if A is a singl. and $x \neq x_o$

Features

(14) $f(x) \doteq y \mathbin{\&} f(x) \doteq z \mathbin{\&} C \;\rightarrow_{x_o}\; [z/y](f(x) \doteq y \mathbin{\&} C)$ if $z \neq x_o$

(15) $f(x) \doteq y \mathbin{\&} \neg f(x) \mathbin{\&} C \;\rightarrow_{x_o}\; x_o{:}\,\bot$

Equations

(16) $x \doteq x \mathbin{\&} C \;\rightarrow_{x_o}\; C$

(17) $x \doteq y \mathbin{\&} C \;\rightarrow_{x_o}\; [x/y]C$ if $x \neq x_o$

(18) $x \doteq y \mathbin{\&} C \;\rightarrow_{x_o}\; [y/x]C$ if $y \neq x_o$

(19) $x \not\doteq x \mathbin{\&} C \;\rightarrow_{x_o}\; x_o{:}\,\bot$

Figure 5: The simplification rules for sorts, features and equations.

6. if $f(x) \doteq y$ is in C, then $\neg f(x)$ is not in C.

The *dependency relation* $\rightarrow_C$ of a feature clause C is a binary relation on the set of all variables defined as follows:

$$x \rightarrow_C y \quad :\Longleftrightarrow \quad \exists\,(xp \doteq y) \in C \;\text{ or }\; \exists\,(x \doteq y) \in C \;\text{ or } \\ \exists\,(f(x) \doteq y) \in C.$$

A variable x is called a *root* of a feature clause C if $x \rightarrow_C^* y$ for every variable y occurring in C. A set description $x|C$ is called *connected* if x is a root of C.

A set description $x|C$ is called *normal* if it is connected and C is a normal feature clause. In the next section we will show that a normal set description is consistent if and only if it has not the form $x|\{x\colon \bot\}$.

Figures 4 and 5 show the simplification rules for set descriptions. Don't be shocked that there are so many—each rule is actually very simple. First note that we use $c \,\&\, C$ to denote the union $\{c\} \cup C$, where C is supposed to be a feature clause *not* containing the constraint c. The variable x_o decorating the simplification arrow $\rightarrow_{x_o}$ is supposed to be the root variable of the set description that is being simplified. With $[x/y]C$ we denote the feature clause obtained from the feature clause C by replacing every occurrence of the variable x with the variable y. The rules for unfolding containments and eliminating path equations in Figure 4 introduce new variables, that is, variables that don't occur in the clause left of the simplification arrow $\rightarrow_{x_o}$ and that are different from the root variable x_o. The following theorem states the major properties of the simplification rules.

Theorem 3.2 *[Simplification]*

1. *(Invariance) If $x_o|C$ is a set description and $C \rightarrow_{x_o} D$, then $x_o|D$ is an equivalent set description; furthermore, if $x_o|C$ is connected, then $x_o|D$ is connected.*

2. *(Completeness) To every set description whose feature clause is not normal one of the simplification rules in Figure 4 or 5 applies.*

3. *(Termination) There are no infinite chains $C_1 \rightarrow_{x_o} C_2 \rightarrow_{x_o} \cdots$.*

Proof. To show the invariance claim, we have to show that, for every rule, (1) x_o is maintained as a root variable and (2) the denotation stays invariant in every interpretation. This verification is tedious (since there are so many rules) but rather straightforward.

The verification of the completeness claim is straightforward.

To prove the termination claim, we define the complexity of a clause C as the triple $(|C|_1, |C|_2, |C|_3)$, where the component complexities are natural numbers defined as follows:

- $|C|_1 := \sum_{\substack{(x:s)\in C \\ s\neq -}} |s|$, where the size $|s|$ of a feature term s is defined as one would expect
- $|C|_2 := \sum_{(xp\doteq y)\in C} (|p|+1)$, where $|p|$ is the length of the path p
- $|C|_3$ is the number of constraints in C.

The lexical order induced by the canonical order on the natural numbers is a well-founder order on these complexity triples. Since every simplification rule reduces the complexity with respect to this order, we know that the length of a $\rightarrow_{x_o}$-derivation issuing from $x_o|C$ is polynomially bounded in the size of C. □

Corollary 3.3 (Simplification of Set Descriptions) *For every connected set description one can compute in polynomial time an equivalent normal set description.*

Corollary 3.4 *For every simple feature term one can compute in polynomial time an equivalent normal set description.*

The relational language suggests that feature logic can be expressed in standard predicate logic by modeling sorts as unary and features as binary predicates. (Features cannot be modeled as functions since in predicate logic functions are interpreted as total functions.) Several axioms are necessary to restrict the possible interpretations to the interpretations admissible for feature logic. The functionality of a feature f can be expressed by the axiom

$$f(x,y) \wedge f(x,z) \rightarrow y \doteq z.$$

The emptyness of the denotation of $\bot$ can be expressed by

$$\neg\bot(x).$$

For every triple A, B, C such that C is the greatest common subsort of A and B we need the axiom

$$A(x) \wedge B(x) \leftrightarrow C(x)$$

to express that C denotes the intersection of A and B. For every singleton symbol A we need the axioms

$$\exists x.A(x), \qquad A(x) \wedge A(y) \rightarrow x \doteq y.$$

Finally, for every singleton symbol A and every feature symbol f we need the axiom

$$\neg(A(x) \wedge f(x,y))$$

to express that f is not defined on $\mathcal{A}$.

4 Canonical Interpretations

An interpretation $\mathcal{A}$ is called *canonical* if for every two feature terms s and t the following conditions are satisfied:

1. s is consistent if and only if $[\![s]\!]^{\mathcal{A}} \neq \emptyset$
2. s and t are equivalent if and only if $[\![s]\!]^{\mathcal{A}} = [\![t]\!]^{\mathcal{A}}$
3. s is subsumed by t if and only if $[\![s]\!]^{\mathcal{A}} \subseteq [\![t]\!]^{\mathcal{A}}$.

Theorem 4.1 *An interpretation $\mathcal{A}$ is canonical if $[\![x|C]\!]^{\mathcal{A}}$ is nonempty for every normal set description $x|C$ such that C is not $\{x\colon \bot\}$.*

Proof. From the Reduction Proposition in Section 2 we know that an interpretation is canonical if it is canonical for the consistency of feature terms. Since every feature term is equivalent to a finite union of simple feature terms (Disjunctive Normal Form Proposition), we even know that an interpretation is canonical if it is canonical for the consistency of simple feature terms.

In Section 3 we proved that for every simple feature term s one can compute a normal set description $x|C$ such that s and $x|C$ denote the same set in every interpretation. Hence an interpretation is canonical if it is canonical for the consistency of normal set descriptions.

Since a set description $x|\{x\colon \bot\}$ denotes the empty set in every interpretation, the requirement that $[\![x|C]\!]^{\mathcal{A}}$ is nonempty if C is not $\{x\colon \bot\}$ implies that the interpretation $\mathcal{A}$ is canonical. □

In this section, we will construct a family of canonical interpretations (one for every signature) for feature logic. The elements of these canonical interpretations are called "feature structures" and can be depicted as finite, directed graphs with labeled nodes and edges (see Figure 6 for an example). Our feature structures generalize the feature structures used by Rounds and Kasper [21] by providing for cycles and nonsingleton sorts.

There are many possibilities for the formal definition of feature structures. For instance, Aït-Kaci [2] uses rational tree domains and Rounds and Kasper [21] use finite automata. Here, yet another formalization is technically most convenient: we will define feature structures as equivalence classes of normal set descriptions containing only "positive" constraints.

A *positive constraint* is a constraint having either the form $f(x) \doteq y$ or the form $x\colon A$, where A is a sort symbol distinct from $\bot$ and $\top$. A *quasi-feature structure* is a normal set description containing only positive constraints.

Quasi-feature structures do not yet model singletons as singletons since they contain too much syntactic structure. For instance, if A is a singleton symbol and x and y are distinct variables, then $x|\{x\colon A\}$ and $y|\{y\colon A\}$ are distinct quasi-feature structures. This technical complication can be resolved by introducing an equivalence relation that identifies quasi-feature structures that should be equal.

A variable x occurring in a quasi-feature structure $y|C$ is called a *singleton variable* of $y|C$ if C contains a constraint $x\colon A$ such that A is a singleton symbol. We say that two quasi-feature structures are *equivalent* if they are equal up to renaming of singleton variables. If $x|C$ is a quasi-feature structure, we denote the equivalence class containing $x|C$ by $\overline{x|C}$.

Now we define a *feature structure* to be an equivalence class of quasi-feature structures. Figure 6 shows a quasi-feature structure together with a matrix and graph representation of the corresponding feature structure.

Let C be a normal feature clause. Then we use C/x to denote the greatest subset of C such that x is a root of C/x. If $\overline{x|C}$ is a feature structure such

$$
\begin{array}{ll}
X \mid \{ & X\text{:}\,\textbf{sentence}, \\
& \mathit{subj}(X) \doteq S, \\
& \mathit{argree}(X) \doteq A,\ A\text{:}\,\textbf{agreement}, \\
& \mathit{pred}(X) \doteq V,\ \mathit{agree}(V) \doteq A,\ \mathit{syncat}(V) \doteq C, \\
& \mathit{first}(C) \doteq S,\ \mathit{rest}(C) \doteq L,\ L\text{:}\,\mathrm{lambda}\ \}
\end{array}
$$

$$
\left[\begin{array}{l}
X \\
\textbf{sentence} \\
\mathit{subj}\text{:}\,S \\
\mathit{agree}\text{:}\,[A\ \textbf{agreement}] \\
\\
\mathit{pred}\text{:}\ \left[\begin{array}{l} V \\ \mathit{agree}\text{:}\,A \\ \mathit{syncat}\text{:}\,[C\ \mathit{first}\text{:}\,S\ \mathit{rest}\text{:}\,\mathrm{lambda}] \end{array}\right]
\end{array}\right]
$$

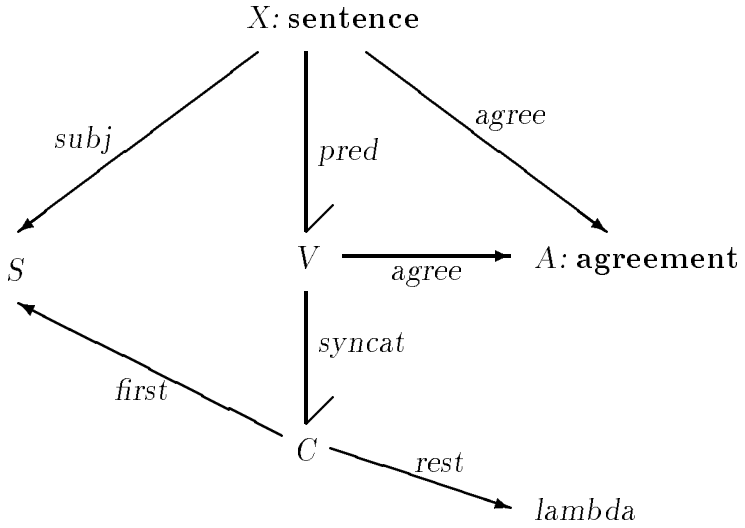

Figure 6: A quasi-feature structure together with a matrix and graph representation of the corresponding feature structure. Variables are written as capital letters. The symbol lambda is a singleton sort and **sentence** and **agreement** are nonsingleton sorts.

that C contains the constraint $f(x) \doteq y$, then $\overline{(y|C/y)}$ is the sub-feature structure reachable through the edge f.

Construction 4.2 (Feature Structure Interpretation $\mathcal{F}$) The set of all feature structures constitutes an interpretation $\mathcal{F}$ defined as follows:

- $\top^{\mathcal{F}}$ is the set of all feature structures
- $A^{\mathcal{F}} := \{(\overline{x|C}) \in \top^{\mathcal{F}} \mid \exists\ (x{:}B) \in C.\ \ B \leq A\}$
- $f^{\mathcal{F}}(\overline{x|C}) := \overline{y|C/y}$ if $(f(x) \doteq y) \in C$.

We will now show that $\mathcal{F}$ is a canonical interpretation.

Theorem 4.3 *Let $x|C$ be a normal set description such that C is not the inconsistent clause $\{x{:}\bot\}$. Then $(\overline{x|C^+}) \in [\![x|C]\!]^{\mathcal{F}}$, where C^+ is the set of all positive constraints in C.*

Proof. Let $x_o|C$ be a normal set description such that C is not the inconsistent clause $\{x_o{:}\bot\}$. Then $\alpha(x) := \overline{x|C^+/x}$ defines an $\mathcal{F}$-assignment α such that $\alpha(x_o) = \overline{x_o|C^+}$. We have to show that $\mathcal{F}, \alpha \models C$.

Suppose $x{:}A$ is in C. Then $x{:}A$ is also in C^+/x. Hence $\alpha(x) = \overline{x|C^+/x} \in A^{\mathcal{F}}$.

Suppose $x{:}\neg A$ is in C. We have to show that $\alpha(x) \notin A^{\mathcal{F}}$. Suppose $\alpha(x) = \overline{x|C^+/x} \in A^{\mathcal{F}}$. Then C must contain a containment $x{:}B$ such that $B \leq A$, which is a contradiction since C is normal and contains $x{:}\neg A$.

Suppose $\neg f(x)$ is in C. We have to show that $f^{\mathcal{F}}$ is not defined on $\alpha(x)$. Suppose $f^{\mathcal{F}}$ is defined on $\alpha(x) = \overline{x|C^+/x}$. Then C must contain a feature equation $f(x) \doteq y$, which is a contradiction since C is normal and contains $\neg f(x)$.

Suppose $f(x) \doteq y$ is in C. Then $f(x) \doteq y$ is in C^+/x. Hence $f^{\mathcal{F}}(\alpha(x)) = f^{\mathcal{F}}(\overline{x|C^+/x}) = \overline{y|(C^+/x)/y} = \overline{y|C^+/y} = \alpha(y)$.

Suppose $x \not\doteq y$ is in C, where x and y are distinct variables. We have to show that $\alpha(x) = \overline{x|C^+/x} \neq \overline{y|C^+/y} = \alpha(y)$. If x and y are not both singleton variables, this disequation obviously holds. If x and y are both singleton variables, the disequation also holds since x and y must be qualified with different singletons because C is normal. □

Corollary 4.4 *The feature structure interpretation $\mathcal{F}$ is canonical.*

Theorem 4.5 *Deciding the consistency of feature terms is a problem in NP.*

Proof. We have shown in Section 2 that every feature term can be rewritten in polynomial time to an equivalent feature term containing only simple complements. Next we can eliminate all unions by replacing them nondeterministically by one of their arguments. The original feature term is consistent if and only if we can obtain in this way a consistent simple feature term. In Section 3 we have shown that for simple feature terms we can compute in polynomial time an equivalent normal set description. This completes the argument since we know by the preceding theorem that the consistency of normal set descriptions can be decided in constant time. □

Theorem 4.6 *Deciding the consistency of feature terms without complements, agreements and disagreements is an NP-hard problem.*

Proof. The proof uses a reduction given by Kasper [11], which reduces the satisfiability problem for propositional formulas in conjunctive normal form, which is known to be NP-complete, to the consistency problem for feature terms. We assume a signature that has two singleton symbols yes and no and two feature symbols f and $\bar{f}$ for every propositional variable f. Now let ϕ be a propositional formula in conjunctive normal form. Then ϕ can be translated into a feature term s_ϕ by replacing the conjunctions with intersections, the disjunctions with unions, every negated propositional variable $\neg f$ with $\bar{f}$:yes, and every unnegated propositional variable f with f:yes. Furthermore, for every propositional variable f occurring in ϕ let s_f be the feature term

$$s_f := (f\colon \mathrm{yes} \sqcap \bar{f}\colon \mathrm{no}) \sqcup (f\colon \mathrm{no} \sqcap \bar{f}\colon \mathrm{yes}).$$

Now it is easy to verify that ϕ is satisfiable if and only if the feature term

$$s_\phi \sqcap s_{f_1} \sqcap \cdots \sqcap s_{f_n}$$

is consistent, where $f_1, \ldots, f_n$ are the propositional variables occurring in ϕ. □

Corollary 4.7 *Deciding the consistency of feature terms is an NP-complete problem.*

As mentioned before, our feature structures extend the feature structures of Rounds and Kasper [21] by accommodating nonsingleton sorts and cycles. Since the agreement $\epsilon \downarrow f$ denotes a nonempty set in $\mathcal{F}$, cycles are in fact necessary to obtain a canonical interpretation. However, if we only admit interpretations satisfying the "finiteness" condition

$$\forall\, a \in \top^{\mathcal{A}}. \quad \{p \mid a \in \mathcal{D}_p^{\mathcal{A}}\} \text{ is finite,}$$

then feature structures without cycles constitute a canonical interpretation.

Let us call a feature structure *complete* if each of its terminal nodes is a singleton. The feature structure in Figure 6 is not complete since the terminal nodes A and S are variables. One can show that complete feature structures constitute a canonical interpretation, if the signature has at least one singleton and one feature symbol.

Even complete feature structures are not a natural data structure since they still contain redundant syntactic structure. For instance, the feature structures

$$\overline{x|\{f(x) \doteq z,\ z{:}A\}} \quad \text{and} \quad \overline{y|\{f(y) \doteq z,\ z{:}A\}}$$

are distinct if the root variables x and y are distinct. The records outlined in Section 1 do not have this redundancy. Incidentally, Pereira and Shieber [18] formalize feature structures as possibly infinite records.

In general, record structures don't constitute a canonical interpretation. To see this, consider a signature that has one singleton A, two features f and g, and no other symbols. Then there is no record structure that satisfies the consistent feature term

$$f \uparrow g \ \sqcap\ f{:}(f{:}A \sqcap g{:}A) \ \sqcap\ g{:}(f{:}A \sqcap g{:}A).$$

5 Feature Terms with Variables

In this section we will generalize feature terms by accommodating variables and quantification. Using variables agreements can be expressed exponentially more succinct. In particular, with variables one can express nonlocal paths [12], which are used, for instance, in Functional Unification Grammar [14]. In Aït-Kaci's [1, 2] formalism agreements are also expressed with variables.

We will show that every feature term with variables and quantification can be translated into an equivalent feature term without variables. Thus variables

and quantification do not enable us to describe more sets, but just provide for more succinct descriptions, which is of crucial importance for efficient unification algorithms [12, 6]. We will show that the consistency problem for feature terms with variables but without quantifications is still in NP. However, if quantifications are present, we do not know whether the consistency problem remains in NP.

To accommodate variables and quantifications, we extend the abstract syntax of feature terms as follows:

$$
\begin{array}{lll}
s, t, u, v & \bot\!\!\rightarrow & A \quad \textit{sort} \\
& | & f\!:\!s \quad \textit{selection} \\
& | & p \downarrow q \textit{agreement} \\
& | & p \uparrow q \textit{disagreement} \\
& | & s \sqcap t \, \textit{intersection} \\
& | & s \sqcup t \, \textit{union} \\
& | & \neg s \quad \textit{complement} \\
& | & x \quad \textit{variable} \\
& | & \sqcup x.s \, \textit{quantification}
\end{array}
$$

Free variables of a feature term are defined as in predicate logic or the lambda calculus. A feature term is *closed* if it has no free variables.

In the following we will refer to the feature terms of Section 2 as feature terms without variables and mean by a feature term a feature term possibly containing variables and quantifications. Thus, although some of the following propositions and theorems read the same as their counterparts in previous sections, they are actually more general.

The *denotation* of a feature term in an interpretation $\mathcal{A}$ under an $\mathcal{A}$-assignment α is a subset of $\top^{\mathcal{A}}$ defined inductively by the following equations:

- $[\![A]\!]^{\mathcal{A}}_{\alpha} = A^{\mathcal{A}}$
- $[\![f\!:\!s]\!]^{\mathcal{A}}_{\alpha} = \{a \in \top^{\mathcal{A}} \mid f^{\mathcal{A}}(a) \in [\![s]\!]^{\mathcal{A}}_{\alpha}\}$
- $[\![p \downarrow q]\!]^{\mathcal{A}}_{\alpha} = \{a \in \mathcal{D}^{\mathcal{A}}_{p} \cap \mathcal{D}^{\mathcal{A}}_{q} \mid p^{\mathcal{A}}(a) = q^{\mathcal{A}}(a)\}$
- $[\![p \uparrow q]\!]^{\mathcal{A}}_{\alpha} = \{a \in \mathcal{D}^{\mathcal{A}}_{p} \cap \mathcal{D}^{\mathcal{A}}_{q} \mid p^{\mathcal{A}}(a) \neq q^{\mathcal{A}}(a)\}$
- $[\![s \sqcap t]\!]^{\mathcal{A}}_{\alpha} = [\![s]\!]^{\mathcal{A}}_{\alpha} \cap [\![t]\!]^{\mathcal{A}}_{\alpha}$
- $[\![s \sqcup t]\!]^{\mathcal{A}}_{\alpha} = [\![s]\!]^{\mathcal{A}}_{\alpha} \cup [\![t]\!]^{\mathcal{A}}_{\alpha}$
- $[\![\neg s]\!]^{\mathcal{A}}_{\alpha} = \top^{\mathcal{A}} \perp [\![s]\!]^{\mathcal{A}}_{\alpha}$

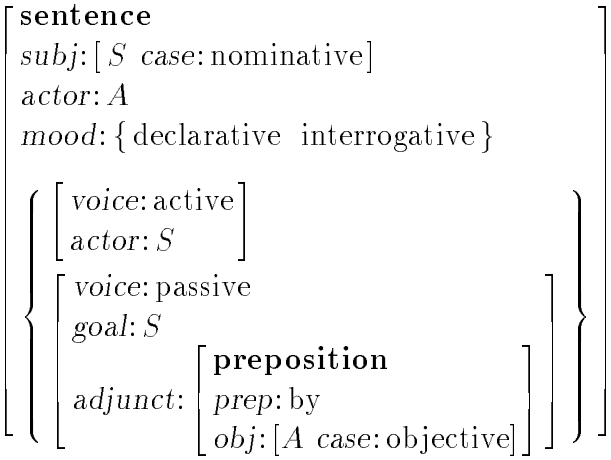

Figure 7: A feature term in matrix notation showing how nonlocal paths can be expressed with variables (here A and S). A conjunctive matrix $[s_1 \ldots s_n]$ stands for $s_1 \sqcap \cdots \sqcup s_n$ and a disjunctive matrix $\{s_1 \ldots s_n\}$ stands for $s_1 \sqcup \cdots \sqcup s_n$.

- $[\![x]\!]^{\mathcal{A}}_{\alpha} = \{\alpha(x)\}$
- $[\![\sqcup x.s]\!]^{\mathcal{A}}_{\alpha} = \bigcup_{a \in \mathsf{T}^{\mathcal{A}}} [\![s]\!]^{\mathcal{A}}_{\alpha[x \leftarrow a]}$.

The *updated assignment* $\alpha[x \leftarrow a]$ is obtained from α by mapping x to a rather than to $\alpha(x)$.

The *denotation* $[\![s]\!]^{\mathcal{A}}$ of a feature term s in an interpretation $\mathcal{A}$ is defined as follows:

$$[\![s]\!]^{\mathcal{A}} = \bigcup_{\substack{\alpha \text{ is an} \\ \text{A-assignment}}} [\![s]\!]^{\mathcal{A}}_{\alpha} .$$

Note that a quantification $\sqcup x.s$ corresponds to an existential quantification in a relational language.

Figures 7 and 8 show two examples for feature terms with variables, the Example in Figure 7 is taken from [12].

Explicit quantification is a concept not present in other feature formalisms. As long as the formalism doesn't offer complements or negations, explicit quantification is superfluous since then all quantifiers can be lifted to the top by renaming variables as necessary. However, if we want to obtain the complement of the denotation of a feature term with variables, we first have to

$$\left[\begin{array}{ll} f\colon\{f\colon x & g\colon\top\} \\ g\colon\{f\colon\top & g\colon x\} \end{array}\right] \quad \doteq \quad \left\{\begin{array}{ll} [f\colon f\colon\top & g\colon f\colon\top] \\ ff\downarrow gg & \\ [f\colon g\colon\top & g\colon f\colon\top] \\ [f\colon g\colon\top & g\colon g\colon\top] \end{array}\right\}$$

Figure 8: A feature term with a coreference that cannot be expressed with local paths. The variable-free feature term to the right is equivalent.

close it by quantifying over its free variables before applying the complement operator.

Example 5.1 Let $\mathcal{A}$ be an interpretation whose domain has at least two elements. Then one verifies easily that $[\![\neg f\colon x]\!]^{\mathcal{A}} = \top^{\mathcal{A}}$. Hence

$$[\![\neg(f\colon x \sqcap g\colon x)]\!]^{\mathcal{A}} = [\![\neg f\colon x \sqcup \neg g\colon x]\!]^{\mathcal{A}} = [\![\neg f\colon x]\!]^{\mathcal{A}} \sqcup [\![\neg g\colon x]\!]^{\mathcal{A}} = \top^{\mathcal{A}}.$$

This shows that without quantification set complements cannot be expressed in the obvious way. However, using quantification, we have

$$\begin{aligned} [\![\neg \sqcup x.(f\colon x \sqcap g\colon x)]\!]^{\mathcal{A}} &= [\![\neg f \downarrow g]\!]^{\mathcal{A}} \\ &= [\![\neg f\colon\top \sqcup \neg g\colon\top \sqcup f \uparrow g]\!]^{\mathcal{A}} \\ &= [\![\neg f\colon\top \sqcup \neg g\colon\top \sqcup (f\colon x \sqcap g\colon \neg x)]\!]^{\mathcal{A}}. \end{aligned}$$

Proposition 5.2 (Tautologies) *Let s and t be feature terms and $u \doteq v$ be an instance of one of the tautologies in Figure 3. If t is obtainable from s by replacing a subterm u with v, then $[\![s]\!]^{\mathcal{A}}_{\alpha} = [\![t]\!]^{\mathcal{A}}_{\alpha}$ for every interpretation $\mathcal{A}$ and every $\mathcal{A}$-assignment α.*

Proposition 5.3 (Distributivity of Quantification) *Let $s_1, \ldots, s_n$ be feature terms and x be a variable. Then*

$$[\![\sqcup x.(s_1 \sqcup \cdots \sqcup s_n)]\!]^{\mathcal{A}}_{\alpha} = [\![(\sqcup x.s_1) \sqcup \cdots \sqcup (\sqcup x.s_n)]\!]^{\mathcal{A}}_{\alpha}$$

for every interpretation $\mathcal{A}$ and every $\mathcal{A}$-assignment α.

A feature term is called *simple* if it contains no union, no quantification, and every contained complement has one of the following forms: $\neg(f\colon\top)$, $\neg x$, or $\neg A$, where A is neither $\bot$ nor $\top$.

Proposition 5.4 (Disjunctive Normal Form) *For every feature term s containing no quantifications one can compute finitely many simple feature terms $s_1, \ldots, s_n$ such that $[\![s]\!]^{\mathcal{A}}_{\alpha} = [\![s_1 \sqcup \cdots \sqcup s_n]\!]^{\mathcal{A}}_{\alpha}$ for every interpretation $\mathcal{A}$ and every $\mathcal{A}$-assignment α.*

Proof. This can be done in the same way it is done in Section 2 for feature terms not containing variables. □

An interpretation is *infinite* if its domain is infinite. Note that the feature structure interpretation $\mathcal{F}$ is infinite since there are infinitely many variables and $x|\emptyset$ and $y|\emptyset$ are distinct feature structures if x and y are distinct variables.

Consistency, subsumption and equivalence of general feature terms are defined as in Section 2, except that we admit from now on only infinite interpretations. This doesn't change anything for feature terms without variables, since the canonical interpretation $\mathcal{F}$ is infinite. The reason for insisting on infinite interpretations is given by the next lemma.

We use $\mathcal{V}(s)$ to denote the set of all variables contained in a feature term s.

Lemma 5.5 *Let s be a simple feature term and x be a variable. Then one can compute in polynomial time a simple feature term t such that $\mathcal{V}(t) = \mathcal{V}(s) \perp \{x\}$ and $[\![\sqcup x.s]\!]^{\mathcal{A}}_{\alpha} = [\![t]\!]^{\mathcal{A}}_{\alpha}$ for every infinite interpretation $\mathcal{A}$ and every $\mathcal{A}$-assignment α.*

Proof. We start by defining the sets $\Pi^+_x(s)$ and $\Pi^-_x(s)$ of positive and negative paths to a variable x in a simple feature term s:

$$\begin{array}{ll}
\Pi^+_x(s) := \emptyset \text{ if } x \notin \mathcal{V}(s) & \Pi^-_x(s) := \emptyset \text{ if } x \notin \mathcal{V}(s) \\
\Pi^+_x(f\!:\!s) := \{fp \mid p \in \Pi^+_x(s)\} & \Pi^-_x(f\!:\!s) := \{fp \mid p \in \Pi^-_x(s)\} \\
\Pi^+_x(s \sqcap t) := \Pi^+_x(s) \cup \Pi^+_x(t) & \Pi^-_x(s \sqcap t) := \Pi^-_x(s) \cup \Pi^-_x(t) \\
\Pi^+_x(x) := \{\epsilon\} & \Pi^-_x(x) := \emptyset \\
\Pi^+_x(\neg x) := \emptyset & \Pi^-_x(\neg x) := \{\epsilon\}
\end{array}$$

Now let s be a simple feature term, x be a variable, $\mathcal{A}$ be an infinite interpretation, and α be an $\mathcal{A}$-assignment. Obtain u from s by first replacing every subterm $\neg x$ with $\top$ and then replacing every remaining x with $\top$. Now we distinguish two cases:

1. $\Pi^+_x(s) = \emptyset$. Then $[\![\sqcup x.s]\!]^{\mathcal{A}}_{\alpha} = [\![u]\!]^{\mathcal{A}}_{\alpha}$ since $\mathcal{A}$ is infinite. To see this note that

$$\bigcup_{a \in M} (M \perp \{a\})^n = M^n$$

for every set M having at least $n+1$ elements.

2. $\Pi_x^+(s) = \{p_i\}_{i=1}^m$, where $m \geq 1$. Let $\Pi_x^-(s) = \{q_i\}_{i=1}^n$ and define

$$t := (u \ \sqcap \ p_1 \downarrow p_2 \sqcap \cdots \sqcap p_1 \downarrow p_m \ \sqcap \ p_1 \uparrow q_1 \sqcap \cdots \sqcap p_1 \uparrow q_n).$$

Then $[\![\sqcup x.s]\!]_\alpha^{\mathcal{A}} = [\![t]\!]_\alpha^{\mathcal{A}}$. □

Theorem 5.6 *For every simple feature term one can compute in polynomial time an equivalent simple feature term not containing variables.*

Proof. Follows immediately from the preceding lemma. □

Theorem 5.7 (NP-Completeness) *Deciding the consistency of feature terms without quantifications is an NP-complete problem.*

Proof. Follows by the Disjunctive Normal Form Proposition, the preceding theorem, and the NP-Completeness Theorem for feature terms without variables. □

Theorem 5.8 (Translation) *For every feature term (possibly containing variables and quantifications) one can compute an equivalent feature term not containing variables.*

Proof. Let s be a feature term. Without loss of generality, we can assume that s contains no free variables. If s contains no quantification, then the claim is trivial. Otherwise it suffices to show that we can eliminate one quantification. Let $\sqcup x.t$ be a subterm of s such that t contains no quantification. Then, by rewriting t to disjunctive normal form, we can compute finitely many simple terms $t_1, \ldots, t_n$ such that

$$\begin{aligned} [\![\sqcup x.t]\!]_\alpha^{\mathcal{A}} &= [\![\sqcup x.(t_1 \sqcup \cdots \sqcup t_n)]\!]_\alpha^{\mathcal{A}} \\ &= [\![(\sqcup x.t_1) \sqcup \cdots \sqcup (\sqcup x.t_n)]\!]_\alpha^{\mathcal{A}} \end{aligned}$$

for every interpretation $\mathcal{A}$ and every $\mathcal{A}$-assignment α. Now the claim follows by the preceding lemma. □

Corollary 5.9 (Canonicity) *The feature structure algebra $\mathcal{F}$ is canonical for feature terms (possibly containing variables and quantifications).*

Proof. We already know that $\mathcal{F}$ is canonical for feature terms not containing variables. Thus we know by the preceding translation theorem that it is canonical for general feature terms. □

Corollary 5.10 (Decidability) *Consistency and subsumption of feature terms (possibly containing variables and quantifications) is decidable.*

Since our quantifier elimination procedure performs a stepwise transformation to disjunctive normal form, it can't be used for a nondeterministic polynomial consistency test. We do not know whether deciding the consistency of general feature terms is still in NP.

We can extend our relational language by allowing for containment constraints $x{:}\,s$, where s is a simple feature term possibly containing variables. The semantics of these generalized containments is defined by

$$\mathcal{A}, \alpha \models x{:}\,s \;:\Longleftrightarrow\; \alpha(x) \in [\![s]\!]^{\mathcal{A}}_{\alpha}.$$

Proposition 5.11 *Let s be a simple feature term and x be a variable not occurring in s. Then $x|\{x{:}\,s\}$ is a set description such that $[\![s]\!]^{\mathcal{A}} = [\![x|\{x{:}\,s\}]\!]^{\mathcal{A}}$ for every interpretation $\mathcal{A}$.*

Next we add two further simplification rules to deal with variables occurring in simple feature terms:

- $x{:}\,y \;\&\, C \;\rightarrow_{x_o}\; x \doteq y \;\&\, C$
- $x{:}\,\neg y \;\&\, C \;\rightarrow_{x_o}\; x \not\doteq y \;\&\, C$.

With these rules Theorem 3.2 generalizes to set descriptions containing simple feature terms with variables.

This gives us a second polynomial time translation of simple feature terms to equivalent normal set descriptions. We will now close the loop and show that every normal set description can be translated into an equivalent simple feature term using the following rules:

1. $x \not\doteq y \ \& C \stackrel{\mathrm{T}}{\perp\!\!\!\rightarrow}_{x_o} x\colon \neg y \ \& C$
2. $f(x) \doteq y \ \& C \stackrel{\mathrm{T}}{\perp\!\!\!\rightarrow}_{x_o} x\colon (f\colon y) \ \& C$
3. $x\colon s \ \& x\colon t \ \& C \stackrel{\mathrm{T}}{\perp\!\!\!\rightarrow}_{x_o} x\colon s \sqcap t \ \& C$
4. $x_o\colon s \ \& y\colon t \ \& C \stackrel{\mathrm{T}}{\perp\!\!\!\rightarrow}_{x_o} x_o\colon s[\pi \leftarrow y \sqcap t] \ \& C \quad$ if $s/\pi = y$.

   Rule (4) replaces an occurrence of y in s with the term $y \sqcap t$.

Proposition 5.12 *If $x_o|C$ is a connected set description and $C \stackrel{\mathrm{T}}{\perp\!\!\!\rightarrow}_{x_o} D$, then $x_o|D$ is an equivalent connected set description.*

Proposition 5.13 *Let $x_o|C$ be a normal set description. Then, using the translation rules, one can compute a simple feature term s such that $C \stackrel{\mathrm{T}}{\perp\!\!\!\rightarrow}_{x_o} x_o\colon s$ and $x_o|C$ and $x_o \sqcap s$ are equivalent.*

With this proposition we can strengthen Theorem 4.1 to:

Theorem 5.14 *An interpretation $\mathcal{A}$ is canonical if and only if $[\![x|C]\!]^{\mathcal{A}}$ is nonempty for every normal set description $x|C$ such that C is not the inconsistent clause $\{x\colon \bot\}$.*

6 Discussion

We have presented a logic for describing objects in domains accessible through a sort lattice and a collection of features. The fact that in knowledge representation the domain of discourse cannot be specified completely (how would you specify humans?) is accounted for by an open world semantics. No single world is fixed but an entire class of admissible worlds is considered. Moreover, the descriptions of the logic do not attempt to specify objects completely. Instead, they consist of constraints that can be satisfied by more than one object. These two sources of indetermination, which are both present in predicate logic, account for the fact that we have only partial information about the world and its objects.

Which worlds are admissible is specified by a signature postulating a classification scheme (the sort lattice) by means of which the following assumptions can be made:

- sort A is a singleton on which no feature is defined
- sort A is a subset of sort B
- sorts A and B are disjoint (make $\bot$ the greatest common subsort of A and B)
- sort C is the intersection of sorts A and B (make C the greatest common subsort of A and B).

Clearly, our signatures are a rather weak mechanism for constraining the admissible worlds. For instance, we cannot make assumptions like

- feature f is not defined on sort A ($(f{:}\top \sqcap A) \leq \bot$)
- feature f is defined on every element of sort A ($A \leq f{:}\top$)
- for every element of sort A, on which feature f is defined, f yields an element of sort B ($A \leq (f{:}B \sqcup \neg f{:}\top)$).

However, these assumptions about admissible worlds can be expressed if we allow for inclusional axioms of the form $s \leq t$. Admitting inclusional axioms in general results in an undecidable logic, but the special forms needed to formulate the above assumptions about features are weak enough to preserve decidability.

Feature logic is closely related to the knowledge representation language KL-ONE [4, 15]. Both formalisms enjoy an open worlds semantics, are based on a classification scheme (the primitive concepts of KL-ONE are sorts) and have set denoting terms (called concept terms in KL-ONE). KL-ONE is more general in that it generalizes features, which are partial functions, to roles, which are relations; on the other hand, feature logic has complements, which aren't available in KL-ONE. Furthermore, KL-ONE has a much stronger apparatus for making assumptions about the admissible worlds (the so-called T-Box). This all suggests that merging feature logic and KL-ONE into a more general knowledge representation formalism is an interesting direction for future research.

We have shown that feature structures constitute a canonical interpretation for feature logic. Thus we can view feature logic without loss of generality as a formalism for reasoning about feature structures. If only feature structures are admitted as interpretation, feature logic has a single domain or closed

world semantics. In fact, this approach is taken in the linguistically motivated formalisms of Rounds and Kasper [21] and Johnson [9]. Although the closed world approach is technically okay, it is unsatisfying philosophically since feature structures are again just partial descriptions of the "real" linguistic objects. The open world semantics presented in this paper renders the detour over feature structures superfluous: one can now view feature terms as directly denoting sets of linguistic objects and there is still no need for making precise what linguistic objects are. Incidentally, this view reconciles the positions of Kaplan and Bresnan [10] and Kay [14]: while Kaplan and Bresnan argue for a strict distinction between feature descriptions and feature structures, Kay insists that there are only feature descriptions (which he calls feature structures).

What is a unification method for feature terms? First, a unification method specifies a normal form for feature terms that exhibits inconsistency, that is,

- every feature term is equivalent to a normal feature term
- a normal feature term is consistent if and only if it is not $\bot$.

Second, a unification method provides an algorithm that, given two normal feature terms s and t, computes a normal feature term equivalent to $s \sqcap t$. Of course, the normal form employed by a unification method can rely on a suitable representation of feature terms, which may be quite different from our syntax.

We have presented a unification method for feature terms in this paper: a normal feature term is represented as a finite set of consistent normal set descriptions, where the union of the denotations of the set descriptions is the denotation of the feature term. All inconsistent feature terms are represented as the empty set (of set descriptions). To unify two normal forms $\{x_i|C_i\}_i$ and $\{y_j|D_j\}_j$, we have to simplify the set description $x_i|(x_i \doteq y_j \,\&\, C_i \,\&\, D_j)$ for every i and every j. The set of the thus obtained consistent normal set descriptions is the result of the unification.

Our unification method requires the representation of feature terms in disjunctive normal form, which, in general, causes an exponential blow up in size. For better efficiency it is crucial to avoid expansion to disjunctive normal form as far as possible. Such unification methods have been devised by Kasper [12] and Eisele and Dörre [6] for feature terms without complements and nonsingleton sorts.

What is the relationship between feature terms and ordinary terms? While the terms of predicate logic denote elements of the universe (more precisely,

functions from assignments to elements), feature terms denote subsets of the universe (more precisely, functions from assignments to sets). While ordinary terms are built up by function application, feature terms are built up by selection (the inverse of unary function application), intersection, union and complement. Thus ordinary terms and feature terms are orthogonal concepts that can coexist profitably in a knowledge representation or logic programming language. For instance, Smolka and Aït-Kaci [24] investigate inheritance hierarchies accommodating both kinds of terms and present a unification algorithm combining order-sorted unification with feature unification.

In Section 3 we have sketched how feature logic can be reduced to predicate logic. Technically, this means that feature logic is just a decidable subset of predicate logic. Incidentally, the same holds for KL-ONE. However, to make predicate logic into a better knowledge representation language, it is crucial to furnish it with more application oriented structure. Additional structure, though technically redundant, eases the formalization of knowledge and serves as the basis for specialized inference methods that can be much more efficient than general purpose mechanisms. (Overstating it a little bit, we could say that predicate logic is to knowledge representation what Turing machines are to programming languages.) For instance, if subsorts are added as a distinguished structure to predicate logic, order-sorted unification [27, 28] replaces ordinary unification and leads to smaller search spaces. Feature terms obviously generalize sorts (which are in fact primitive feature terms) and can be integrated by allowing for containments $x{:}\,s$, where s is a feature term possibly containing variables. For inference, feature term unification can then take the place of order-sorted unification (this needs to be worked out, of course).

The integration of Prolog-like logic programming with feature logic seems to be a very promising line of research. In LOGIN [3], which pioneered this approach, feature terms (without singletons, unions and complements) take the place of ordinary terms and feature unification replaces ordinary term unification. Mukai's [17] language CIL is similar but uses constants instead of sorts. While LOGIN is presented without a declarative semantics, Mukai defines a declarative semantics for CIL using a fixed domain of rational records. Mukai also realizes that CIL is an instance of constraint logic programming, an approach that originated with Colmerauer's [5] Prolog-II and was generalized by Jaffar and Lassez [8]. Recent research [7] investigates logic programming based on feature logic and open world semantics. The presence of feature term unions (disjunctions) can diminish the need for backtracking and feature term complements generalize the disequations of

Prolog-II ($x \neq y$ is equivalent to $x : \neg y$).

References

[1] H. Aït-Kaci. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures.* PhD thesis, University of Pennsylvenia, Philadelphia, PA, 1984.

[2] H. Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351, 1986.

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

[5] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the 2nd International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.

[6] A. Eisele and J. Dörre. Unification of disjunctive feature descriptions. In *Proceedings of the 26th Annual Meeting of the ACL, State University of New York at Buffalo*, pages 286–294, Buffalo, New York, 1988.

[7] M. Höhfeld and G. Smolka. Definite relations over constraint languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, Oct. 1988. To appear in the Journal of Logic Programming.

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

[9] M. Johnson. *Attribute-Value Logic and the Theory of Grammar.* CSLI Lecture Notes 16. Center for the Study of Language and Information, Stanford University, CA, 1988.

[10] R. M. 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, MA, 1982.

[11] R. T. Kasper. *Feature Structures: A Logical Theory with Applications to Language Analysis*. PhD thesis, University of Michigan, Ann Arbor, Mich., 1987.

[12] R. T. Kasper. A unification method for disjunctive feature descriptions. In *Proceedings of the 25th Annual Meeting of the ACL, Stanford University*, pages 235–242, Stanford, CA, 1987.

[13] 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.

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

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

[16] M. D. Moshier and W. C. Rounds. A logic for partially specified data structures. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 156–167, München, W. Germany, 1987.

[17] K. Mukai. Anadic tuples in Prolog. Technical Report TR-239, ICOT, Tokyo, Japan, 1987.

[18] F. Pereira and S. M. Shieber. The Semantics of Grammar Formalisms seen as Computer Languages. In *Proceedings of 10th International Conference on Computational Linguistics*, pages 123–129, Stanford, 1984.

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

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

[21] W. C. Rounds and R. T. 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, MA, 1986.

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

[23] G. Smolka. Feature constraint logics for unification grammars. IWBS Report 93, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, November 1989. To appear in the Journal of Logic Programming.

[24] G. Smolka and H. Aït-Kaci. Inheritance hierarchies: Semantics and unification. *Journal of Symbolic Computation*, 7:343–370, 1989.

[25] G. Smolka, W. Nutt, J. A. Goguen, and J. Meseguer. Order-Sorted Equational Computation. In H. Aït-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2, Rewriting Techniques*, chapter 10, pages 297–367. Academic Press, New York, N.Y., 1989.

[26] H. Uszkoreit. From Feature Bundles to Abstract Data Types: New Directions in the Representation and Processing of Linguistic Information. In A. Blaser, editor, *Natural Language at the Computer—Contributions to Syntax and Semantics for Text Processing and Man-Maschine Translation*, pages 31–64. Lecture Notes in Computer Science 320, Springer-Verlag, Berlin, Germany, 1988.

[27] C. Walther. A mechanical solution of Schubert's steamroller by many-sorted resolution. *Artificial Intelligence*, 26:217–224, 1985.

[28] C. Walther. Many-sorted unification. *Journal of the ACM*, 35(1):1–17, January 1988.