Feature Constraint Logics for Unification Grammars

Gert Smolka

German Research Center for Artificial Intelligence and
Universität des Saarlandes
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
smolka@dfki.uni-sb.de

Abstract

This paper studies feature description languages that have been developed for use in unification grammars, logic programming and knowledge representation. The distinctive notational primitive of these languages are features that can be understood as unary partial functions on a domain of abstract objects. We show that feature description languages can be captured naturally as sublanguages of first-order predicate logic with equality and show the equivalence of a loose Tarski semantics with a fixed feature graph semantics for quantifier-free constraints. For quantifier-free constraints we give a constraint solving method and show the NP-completeness of satisfiability checking. For general feature constraints with quantifiers satisfiability is shown to be undecidable. Moreover, we investigate an extension of the logic with sort predicates and set-denoting expressions called feature terms.

Journal of Logic Programming, vol. 12, 1992, pp. 51--87.

Previous version as IWBS Report 93, November 1989, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany.

Contents

1 Introduction

In the last decade a new type of grammar formalism, now commonly referred to as unification grammars, has evolved from research in linguistics, computational linguistics and artificial intelligence. 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.

Conceptually, a unification grammar formalism can be divided into a phrase structure component and a constraint logic. The phrase structure components of some formalisms are given by context-free rules. In these formalisms the context-free phrase structure rules are augmented with constraints taken from the constraint logic. These constraints further confine the derivations licensed by the phrase structure rules. Thus grammatical knowledge can be formulated at two levels, the phrase structure and the constraint level. In practice most of the grammatical knowledge is expressed at the constraint level. Since the phrase structure component provides for inductive definition (or, from the computational point of view, recursion), the constraint logic can be kept decidable.

Two types of constraint logics have been used in unification grammar formalisms. The constraint logic of Definite Clause Grammars [36] is identical with the constraint logic of Prolog and consists of first-order equations interpreted in the free term algebra. The other type of constraint logic, which evolved with the now predominant feature-based unification grammars, is based on the notion of features and has only recently become subject of theoretical investigation and formalization. It is this family of constraint logics that we further establish and investigate in this paper.

In the context of unification grammars, a feature is a functional property or attribute of abstract (linguistic) objects. For instance, the abstract object associated with the sentence

John sings a song

may have the features *subject*, *predicate*, *object* and *tense*. Mathematically, features can be modeled as partial functions that can be applied to abstract objects. If, for instance, the feature *object* is applied to the abstract object representing the above sentence, one obtains a further abstract object representing the object phrase "a song". Primitive abstract objects are atoms like *singular*, *plural* or *present* that don't have features defined on them. Since the relevant properties of abstract objects are determined by the values of the features defined on them, abstract objects can be represented as rooted graphs. The nodes of such a "feature graph" stand for abstract objects and the edges represent the defined features.

Figure 1 shows a feature graph that may represent the abstract object associated with the sentence "John sings a song".[1] 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

[1] How the abstract object associated with a sentence looks is determined by the grammar. Unification grammar formalisms are comparable to programming languages in that the same set of sentences can be specified by many different grammars relying on different linguistic theories.

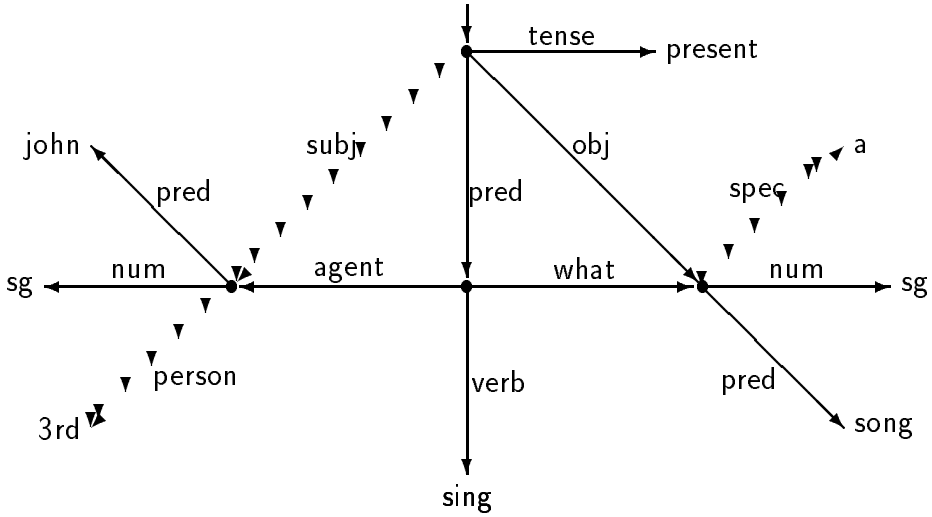

Figure 1: A feature graph.

$$\left[\begin{array}{l} \mathsf{x} \\ \mathit{person} \\ \mathsf{mother}: \left[\begin{array}{l} \mathit{employee} \\ \mathsf{firstchild}: \mathsf{x} \\ \mathsf{secondchild}: \mathsf{y} \\ \mathsf{age}: 54 \end{array}\right] \\ \mathsf{oldestsister}: \mathsf{y} \\ \mathsf{age}: 27 \end{array}\right]$$

Figure 2: A feature term.

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.

Two kinds of feature descriptions have been developed. Bresnan and Kaplan's Lexical-Functional Grammar [17], Shieber's PATR-II formalism [46], and Johnson's Attribute-Value Logic [16] employ boolean combinations of equations built from features (used as unary functions), atoms (used as constants) and variables. Kay's Functional Unification Grammar [24, 25, 26], Aït-Kaci's ψ-Term Calculus [2, 3, 4], and Kasper and Rounds' logic [23, 39] employ set-denoting expressions, called feature terms in this paper, that come in different syntactic guises (Figure 2 gives an example). Feature terms have much in common with the concept descriptions of terminological logics [5, 33, 34] used in knowledge representation. In fact, Aït-Kaci's ψ-term calculus was developed independently from the linguistically oriented approaches for application in Logic Programming and knowledge representation.

This paper shows that both kinds of feature descriptions can be captured as sublanguages of first-order predicate logic with equality. This reduction to a very well-understood framework

is surprisingly natural and brings much simplicity and clarity. This approach is already suggested by Bresnan and Kaplan's pioneering paper on Lexical-Functional Grammar [17] and has been worked out further in Johnson's dissertation [16]. However, the present paper, which is an elaboration of [47], shows for the first time that the feature term descriptions of Kay, Aït-Kaci, and Kasper and Rounds can be embedded as well into predicate logic. It turns out that feature terms are merely a syntactic extension, which can be eliminated in linear time.

In Lexical-Functional Grammar, feature equations are interpreted in a domain of feature graphs (see Figure 1 for an example). In this paper we will admit much more general interpretations called feature algebras. The set of all feature graphs can be arranged to one particular feature algebra, the so-called feature graph algebra. We will show that the feature graph algebra is canonical in that a quantifier-free constraint is satisfiable in some feature algebra if and only if it is satisfiable in the feature graph algebra.

To employ unification grammars for parsing, one needs a solution method for the employed feature constraints. In the case of equational descriptions the solution method can be a constraint solving method, which simplifies a constraint to a certain normal form and thereby determines whether the constraint is satisfiable. For feature terms so-called unification methods have been developed, which compute for two feature terms in normal form a new feature term in normal form combining their information. We show that feature term unification can be reduced to the more general but, nevertheless, technically simpler constraint solving.

It was the unification operation for feature terms conceived by Kay that led to the name unification grammars. Unfortunately, this name is rather misleading since it is derived from an operation that may or may not be employed in an implementation of a unification grammar formalism.

This paper is organized as follows. In Section 2, we outline a simple unification grammar formalism based on context-free phrase structure rules to illustrate the basic ideas and the interaction between phrase structure rules and feature constraints. In Section 3, we introduce the possible interpretations of feature descriptions, which we call feature algebras. Furthermore, we formalize feature graphs and show that they constitute one possible feature algebra, which, as it will turn out in Section 5, enjoys important prototypical properties. In Section 4, we define constraints and start with the development of a constraint solving algorithm. In Section 5, we develop the remaining phase of the constraint solving algorithm and prove several of our main theorems. In Section 6, we extend our formalism to feature terms. In Section 7, we further extend our formalism to sorts. In Section 8, we prove two undecidability results, one concerning quantified constraints and one concerning cyclic sort equations. The final 9th Section relates our approach to previous work and discusses possible extensions.

2 Constraint Grammars

In this section we outline a simple unification grammar formalism based on context-free phrase structure rules and feature equations. Grammars in this formalism will be called

constraint grammars.

To have an example, we will model simple sentences consisting of a subject, a predicate and an object (like "John sings a song"). For this we use the familiar phrase structure rules

$$\begin{aligned} \mathsf{S} &\longrightarrow \mathsf{NP}\ \ \mathsf{VP} \\ \mathsf{NP} &\longrightarrow \mathsf{D}\ \ \mathsf{N} \\ \mathsf{VP} &\longrightarrow \mathsf{V}\ \ \mathsf{NP}. \end{aligned}$$

One phrase structure tree licensed by the rules is shown in Figure 3. Every node of this phrase structure tree comes with a distinct variable. Hence we have the two variables NP and NP_1 for the two noun phrase nodes. *It are the variables associated with the nodes of a phrase structure tree what is constrained by the constraints of the rules.* For our phrase structure rules we may have constraints as follows:

$$\begin{aligned} \mathsf{S} &\longrightarrow \mathsf{NP}\ \mathsf{VP} \\ &\quad\ \mathsf{subj}\ \mathsf{S} \doteq \mathsf{NP} \wedge \mathsf{S} \doteq \mathsf{VP} \\ \mathsf{NP} &\longrightarrow \mathsf{D}\ \mathsf{N} \\ &\quad\ \mathsf{NP} \doteq \mathsf{D} \doteq \mathsf{N} \\ \mathsf{VP} &\longrightarrow \mathsf{V}\ \mathsf{NP} \\ &\quad\ \mathsf{VP} \doteq \mathsf{V} \wedge \mathsf{obj}\ \mathsf{VP} \doteq \mathsf{NP}. \end{aligned}$$

The constraints of the rules define a constraint for every phrase structure tree licensed by the rules. For the phrase structure tree in Figure 3 we obtain the constraint

$$\begin{aligned} \exists \mathsf{NP}\ \exists \mathsf{VP}\ \exists \mathsf{V}\ \exists \mathsf{NP}_1\ \exists \mathsf{D}\ \exists \mathsf{N} \quad & (\mathsf{subj}\ \mathsf{S} \doteq \mathsf{NP} \wedge \mathsf{S} \doteq \mathsf{VP} \wedge \\ & \mathsf{VP} \doteq \mathsf{V} \wedge \mathsf{obj}\ \mathsf{VP} \doteq \mathsf{NP}_1 \wedge \\ & \mathsf{NP}_1 \doteq \mathsf{D} \doteq \mathsf{N}). \end{aligned}$$

The constraints are formulas as in predicate logic. All variables but the root variable are existentially quantified. The feature equation $\mathsf{subj}\ \mathsf{S} \doteq \mathsf{NP}$, for instance, says that NP is the value of the feature subj applied to S. Features are unary partial functions. The variables in the constraints range over abstract objects representing the concrete phrases.

The constraints of the phrase structure rules yield the skeleton in which the constraints coming with the lexical rules are put. Note that the constraint of the phrase structure tree in Figure 3 identifies the variables D and N representing the determiner and noun of the noun phrase NP_1. Hence the constraints for the determiner and noun apply to the same abstract object. With that it is easy to enforce that the determiner and noun are either both in plural or both in singular. For instance, we may have the lexical rules

$$\begin{aligned} \mathsf{D} &\longrightarrow \mathsf{a} \\ &\quad\ \mathsf{spec}\ \mathsf{D} \doteq \mathsf{a}\ \ \wedge \\ &\quad\ \mathsf{num}\ \mathsf{D} \doteq \mathsf{sg} \\ \mathsf{N} &\longrightarrow \mathsf{song} \\ &\quad\ \mathsf{pred}\ \mathsf{N} \doteq \mathsf{song}\ \ \wedge \\ &\quad\ \mathsf{num}\ \mathsf{N} \doteq \mathsf{sg} \end{aligned}$$

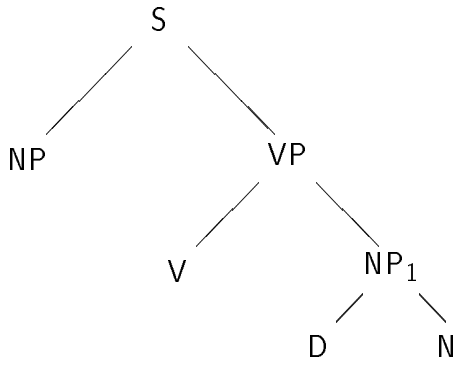

Figure 3: A phrase structure tree.

$$\begin{aligned} \mathsf{N} \quad \longrightarrow \quad & \mathsf{songs} \\ & \mathsf{pred\ N} \doteq \mathsf{songs} \ \wedge \\ & \mathsf{num\ N} \doteq \mathsf{pl}. \end{aligned}$$

Now the noun phrase "a song" yields the satisfiable constraint

$$\begin{gathered} \mathsf{NP}_1 \doteq \mathsf{D} \doteq \mathsf{N} \ \wedge \\ \mathsf{spec\ D} \doteq \mathsf{a} \wedge \mathsf{num\ D} \doteq \mathsf{sg} \ \wedge \\ \mathsf{pred\ N} \doteq \mathsf{song} \wedge \mathsf{num\ N} \doteq \mathsf{sg} \end{gathered}$$

while the ungrammatical noun phrase "a songs" yields the unsatisfiable constraint

$$\begin{gathered} \mathsf{NP}_1 \doteq \mathsf{D} \doteq \mathsf{N} \ \wedge \\ \mathsf{spec\ D} \doteq \mathsf{a} \wedge \mathsf{num\ D} \doteq \mathsf{sg} \ \wedge \\ \mathsf{pred\ N} \doteq \mathsf{songs} \wedge \mathsf{num\ N} \doteq \mathsf{pl}. \end{gathered}$$

The unsatisfiability of the constraint stems from the fact that we assume that the atoms sg and pl are different.

Now consider the lexical entry for the verb sings:

$$\begin{aligned} \mathsf{V} \quad \longrightarrow \quad & \mathsf{sings} \\ & \mathsf{tense\ V} \doteq \mathsf{present} \ \wedge \\ & \mathsf{verb\ pred\ V} \doteq \mathsf{sing} \ \wedge \\ & \mathsf{agent\ pred\ V} \doteq \mathsf{subj\ V} \ \wedge \\ & \mathsf{what\ pred\ V} \doteq \mathsf{obj\ V} \ \wedge \\ & \mathsf{num\ subj\ V} \doteq \mathsf{sg} \ \wedge \\ & \mathsf{person\ subj\ V} \doteq \mathsf{3rd}. \end{aligned}$$

Since the constraints of the phrase structure rules equate the variables S, VP and V of the phrase structure tree in Figure 3, the constraints of the verb directly constrain the entire sentence. This makes good sense since the predicate of a sentence dominates the sentence syntactically and semantically. The tense of the predicate, for instance, is also the tense of

the entire sentence. This arrangement also makes it easy to enforce agreement in numerus and person between the subject and the predicate of the sentence. The last two constraints of the lexical rule for sings prescribe the numerus and the person of the subject of the sentence since we have

$$\begin{aligned}&\mathsf{subj\ S \doteq NP \wedge S \doteq VP\ \wedge\ VP \doteq V\ \wedge}\\&\mathsf{num\ subj\ V \doteq sg \wedge person\ subj\ V \doteq 3rd},\end{aligned}$$

which implies

$$\mathsf{num\ NP \doteq sg \wedge person\ NP \doteq 3rd}.$$

If we add the lexical rule

$$\begin{aligned}\mathsf{NP} \longrightarrow\ &\mathsf{John}\\&\mathsf{pred\ NP \doteq john}\ \wedge\\&\mathsf{num\ NP \doteq sg}\ \wedge\\&\mathsf{person\ NP \doteq 3rd},\end{aligned}$$

we can account for the sentence "John sings a song". Figure 4 shows the phrase structure tree of this sentence whose constraint in its full glory looks as follows:

$$\begin{aligned}&\mathsf{\exists NP\ \exists VP\ \exists V\ \exists NP_1\ \exists D\ \exists N\ (}\\&\mathsf{(subj\ S \doteq NP \wedge S \doteq VP)} \wedge && S \longrightarrow NP\ VP\\&\mathsf{(pred\ NP \doteq john \wedge num\ NP \doteq sg \wedge person\ NP \doteq 3rd)} \wedge && NP \longrightarrow John\\&\mathsf{(VP \doteq V \wedge obj\ VP \doteq NP_1)} \wedge && VP \longrightarrow V\ NP\\&\mathsf{(tense\ V \doteq present} \wedge && V \longrightarrow sings\\&\mathsf{verb\ pred\ V \doteq sing} \wedge\\&\mathsf{agent\ pred\ V \doteq subj\ V} \wedge\\&\mathsf{what\ pred\ V \doteq obj\ V} \wedge\\&\mathsf{num\ subj\ V \doteq sg} \wedge\\&\mathsf{person\ subj\ V \doteq 3rd)}\ \wedge\\&\mathsf{(NP_1 \doteq D \doteq N)} \wedge && NP \longrightarrow D\ N\\&\mathsf{(spec\ D \doteq a \wedge num\ D \doteq sg)} \wedge && D \longrightarrow a\\&\mathsf{(pred\ N \doteq song \wedge num\ N \doteq sg)\ )} && N \longrightarrow song.\end{aligned}$$

This monster is logically equivalent to the more digestible constraint

$$\begin{aligned}\mathsf{\exists S \exists P \exists O(}\ \ &\mathsf{subj\ x \doteq S}\ \wedge\ \mathsf{pred\ x \doteq P}\ \wedge\ \mathsf{obj\ x \doteq O}\ \wedge \mathsf{tense\ x \doteq present}\ \wedge\\&\mathsf{pred\ S \doteq john}\ \wedge\ \mathsf{num\ S \doteq sg}\ \wedge\ \mathsf{person\ S \doteq 3rd}\ \wedge\\&\mathsf{verb\ P \doteq sing}\ \wedge\ \mathsf{agent\ P \doteq S}\ \wedge\ \mathsf{what\ P \doteq O}\ \wedge\\&\mathsf{spec\ O \doteq a}\ \wedge\ \mathsf{num\ O \doteq sg}\ \wedge\ \mathsf{pred\ O \doteq song}\ )\end{aligned}$$

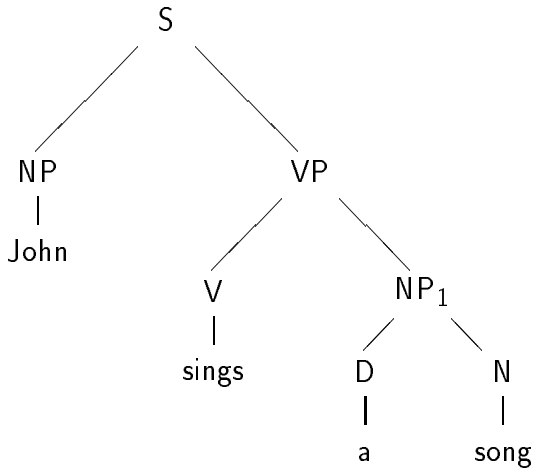

Figure 4: A complete phrase structure tree.

whose graphical representation is the feature graph in Figure 1. This shows another nice property of our grammar formalism: The satisfiable constraint of a phrase structure tree can serve as a logical representation of the corresponding reading of the sentence.

A sequence of words is licensed by a constraint grammar if it has at least one phrase structure tree whose constraint is satisfiable. Given a constraint grammar, we call a phrase structure tree *admissible* if its constraint is satisfiable, and we call a sequence of words a *sentence* if it has at least one admissible phrase structure tree. Typically, a sentence will have several admissible phrase structure trees corresponding to different possible readings of the sentence.

The constraint of a phrase structure tree is defined inductively with respect to the structure of the tree. Hence the constraint of a phrase structure tree can be computed incrementally during the construction of the tree. Since one is interested in the logical meaning of the constraint and not its internal syntactic structure, it suffices to compute the constraint up to logical equivalence. Thus one is free to simplify partial constraints to logically equivalent ones during the construction of a phrase structure tree.

Constraints in our grammar formalism are built from feature equations using conjunction and existential quantification. Such constraints are unsatisfiable if a subconstraint is unsatisfiable. Hence, if the constraint of a partial phrase structure tree turns out to be unsatisfiable, one knows that the partial phrase structure tree cannot be completed to an admissible phrase structure tree. Thus an efficient parser for our constraint grammars will employ an incremental constraint solving method that, given a constraint, simplifies it to a logically equivalent normal form exhibiting unsatisfiability. Since the nonlexical rules combine the constraints of the constituents by conjunction, the constraint solving method can be specialized in that it computes the normal form of the conjunction of two constraints in normal form. In the literature on unification grammars such constraint solving methods are often called unification methods.

The grammar can be written such that the constraint of an admissible phrase structure tree of a sentence is a convenient representation of the corresponding reading of the sentence.

In the simple constraint logic employed so far, every satisfiable constraint can be represented without loss of information as a feature graph, which can serve as a convenient data structure for subsequent, more semantically oriented processing steps of a natural language understanding system.

Obviously, there are some strong analogies between constraint grammar formalisms and (logic) programming languages in that grammar formalisms allow for very different grammars describing the same set of sentences. How a grammar is written depends mainly on the linguistic theory being adhered to, but also on operational properties like efficiency for parsing.

The word problem of a grammar is to decide for a given sequence of words whether it is a sentence of the grammar. One can show that even the simple formalism outlined above allows for grammars having an undecidable word problem by adapting proofs given by Johnson [16] or Rounds and Manaster-Ramer [40] for slightly different formalisms. For grammars employing a decidable constraint logic (that is, it is decidable whether a constraint is satisfiable), the *off-line parsability constraint* [17] is a sufficient condition for the decidability of the word problem. A grammar satisfies the off-line parsability constraint if the number of different phrase structure trees (not necessarily admissible) of a sequence of words is bounded by a computable function of the length of that sequence. The off-line parsability constraint is satisfied, for instance, if the right-hand side of every context-free rule contains either at least one terminal or at least two nonterminals.

If a word has more than one meaning,[2] one can either have several lexical rules for the same word or have only one rule whose constraint is obtained as the disjunction of the constraints describing the different meanings. Sometimes it is also convenient to use logical implication in the constraints of lexical rules. For instance, the constraint for the word "sing" may contain the implication

$$\mathsf{person\ subj\ V} \doteq \mathsf{3rd} \quad \rightarrow \quad \mathsf{num\ subj\ V} \doteq \mathsf{pl},$$

which can be read as: If the subject of the sentence is in third person, then it must be in plural.

The simple unification grammar formalism sketched here bears much resemblance with the PATR formalism developed at SRI International by Shieber and his colleagues [46, 43]. It is also closely related to Bresnan and Kaplan's Lexical-Functional Grammar formalism (LFG) [17]. Other unification grammar formalisms such as Kay's Functional Unification Grammar (FUG) [24, 25, 26, 40], Uszkoreit's Categorial Unification Grammar [49], or Pollard and Sag's HPSG [37] employ different phrase structure rules and different feature constraints.

Shieber's [44] 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 [37] and [35]. Johnson's thesis [16] gives a formal account of an LFG-like formalism and investigates a feature constraint language with disjunctions and negations. Shieber's thesis [45] gives a rigorous formalization of the PATR formalism.

[2] For instance, the word "drink" can be used as transitive verb or as noun.

3 Feature Algebras and Feature Graphs

In this section we define the possible interpretations of the feature descriptions to be discussed. These interpretations are called *feature algebras* and are like the usual interpretations of predicate logic. However, we admit only constants and features as nonlogical symbols, where features are binary predicates that must be interpreted as functional relations.

We assume three pairwise disjoint sets of symbols: **variables** (denoted by x, y, z), **features** (denoted by f, g, h), and **constants** (denoted by a, b, c). Constants are also called **atoms**. We assume that there are infinitely many variables. The letters s and t will always denote variables or atoms.

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 every atom a an element $a^\mathcal{I} \in \mathbf{D}^\mathcal{I}$ and to every feature f a set of ordered pairs $f^\mathcal{I} \subseteq \mathbf{D}^\mathcal{I} \times \mathbf{D}^\mathcal{I}$ such that the following conditions are satisfied:

1. if (d, e) and (d, e') are in $f^\mathcal{I}$, then $e = e'$ *(features are functional)*
2. if $a \neq b$, then $a^\mathcal{I} \neq b^\mathcal{I}$ *(unique name assumption)*
3. if f is a feature and a is an atom, then there exists no $d \in \mathbf{D}^\mathcal{I}$ such that $(a^\mathcal{I}, d) \in f^\mathcal{I}$ *(atoms are primitive)*.

The first condition says that features are interpreted as unary partial functions, and the third condition says that features must not be defined on atoms. We write $f^\mathcal{I}(d) = e$ if and only if $(d, e) \in f^\mathcal{I}$. Furthermore, the **domain** of a feature f in a feature algebra $\mathcal{I}$ is

$$\mathbf{D}(f^\mathcal{I}) \quad := \quad \{d \in \mathbf{D}^\mathcal{I} \mid \exists e \in \mathbf{D}^\mathcal{I} : \ (d, e) \in f^\mathcal{I}\}.$$

Next we define feature graphs (Figure 1 shows an example of a feature graph). The set of all feature graphs will yield a special feature algebra that enjoys prototypical properties comparable to the properties of term algebras in equational logic.

A feature graph is a finite, rooted, connected and directed graph whose edges are labeled with features. For every node, the labels of the edges departing from it must be pairwise distinct. Moreover, every inner node of a feature graph must be a variable, and every terminal node must be either an atom or a variable. Feature graphs can be seen as finite deterministic automata, which is the formalization given by Kasper and Rounds [23, 39]. In contrast to Kasper and Rounds, however, we admit cyclic feature graphs.

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 an atom. A **feature graph** is either a pair $(a, \emptyset)$, where a is an atom 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 and xft are 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.

A feature graph G is called a **subgraph** of a feature graph G' if the root of G is a variable or atom occurring in G' and every edge of G is an edge of G'. The subgraphs of a feature graph G are partially ordered by

$$G' \leq G'' \iff G' \text{ is a subgraph of } G''.$$

If G is a feature graph and s is an atom or variable occurring in G, we use G_s to denote the unique maximal subgraph of G whose root is s.

The **feature graph algebra** $\mathcal{F}$ is defined 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. $(G, G') \in f^{\mathcal{F}}$ if and only if G has an edge xfs such that x is the root of G and $G' = G_s$.

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

Feature graphs can be understood as data structures representing information. There exists a natural preorder on feature graphs, usually called subsumption preorder, that orders feature graphs according to their information content. The subsumption preorder is such that two feature graphs are equivalent if and only if they are equal up to consistent variable renaming.[3] It turns out that the subsumption preorder on feature graphs generalizes to general feature algebras.[4] To this purpose we define the notion of a partial homomorphism between feature algebras.

Let $\mathcal{I}$ and $\mathcal{J}$ be feature algebras. A **partial homomorphism** from $\mathcal{I}$ to $\mathcal{J}$ is a partial function γ from $\mathbf{D}^{\mathcal{I}}$ to $\mathbf{D}^{\mathcal{J}}$ such that

1. if a is an atom and γ is defined on $a^{\mathcal{I}}$, then $\gamma(a^{\mathcal{I}}) = a^{\mathcal{J}}$
2. if f is feature and $f^{\mathcal{I}}$ and γ are defined on $d \in \mathbf{D}^{\mathcal{I}}$, then γ is defined on $f^{\mathcal{I}}(d)$, $f^{\mathcal{J}}$ is defined on $\gamma(d)$, and $f^{\mathcal{J}}(\gamma(d)) = \gamma(f^{\mathcal{I}}(d))$.

A **partial endomorphism** of a feature algebra $\mathcal{I}$ is a partial homomorphism from $\mathcal{I}$ to $\mathcal{I}$.

Let $\mathcal{I}$ be a feature algebra. The **subsumption preorder** $\preceq^{\mathcal{I}}$ of $\mathcal{I}$ is a preorder on $\mathbf{D}^{\mathcal{I}}$ defined as follows:

$$d \preceq^{\mathcal{I}} e \iff \exists \text{ partial endomorphism } \gamma \text{ of } \mathcal{I} \text{ such that } \gamma(d) = e.$$

[3] However, the feature graph algebra $\mathcal{F}$ does not identify feature graphs that are equivalent. In the presence of negative constraints $\mathcal{F}$ would fail to be canonical if equivalent feature graphs were identified (see Section 5).

[4] This was discovered first by Bill Rounds who presented the idea in a seminar he gave in the summer of 1989 at the University of Stuttgart.

If $d \preceq^{\mathcal{I}} e$, we say that d is **more general** than e (in $\mathcal{I}$) or, conversely, that e is **more specific** than d (in $\mathcal{I}$). Furthermore, we say that d and e are **equivalent** (in $\mathcal{I}$) if $d \preceq^{\mathcal{I}} e$ and $e \preceq^{\mathcal{I}} d$.

Next we show that the subsumption preorder $\preceq^{\mathcal{F}}$ of the feature graph algebra $\mathcal{F}$ coincides with the subsumption order on feature graphs given by Kasper and Rounds [23, 39]. To this purpose we introduce paths.

A **path** is a finite, possibly empty sequence of features. The letters p and q will always denote paths. Let $\mathcal{I}$ be a feature algebra and $p = f_n \cdots f_1$ be a path. Then p is interpreted as a unary partial function $p^{\mathcal{I}}$ from $\mathbf{D}^{\mathcal{I}}$ to $\mathbf{D}^{\mathcal{I}}$ defined as follows: If p is empty (that is, $n = 0$), then $p^{\mathcal{I}}$ is the identity function of $\mathbf{D}^{\mathcal{I}}$; otherwise $p^{\mathcal{I}}$ is the composition of the partial functions $f_n^{\mathcal{I}}, \ldots, f_1^{\mathcal{I}}$, where $f_1^{\mathcal{I}}$ is applied first. As with features, we write $p^{\mathcal{I}}(d) = e$ if and only if $(d, e) \in p^{\mathcal{I}}$.

Let $\mathcal{I}$ be a feature algebra and $d \in \mathbf{D}^{\mathcal{I}}$. Then $e \in \mathbf{D}^{\mathcal{I}}$ is called a **component** of d if there exists a path p such that $e = p^{\mathcal{I}}(d)$.

Proposition 3.1 *Let γ be a partial homomorphism from a feature algebra $\mathcal{I}$ to a feature algebra $\mathcal{J}$ and let p be a path. If γ and $p^{\mathcal{I}}$ are defined on $d \in \mathbf{D}^{\mathcal{I}}$, then γ is defined on $p^{\mathcal{I}}(d)$, $p^{\mathcal{J}}$ is defined on $\gamma(d)$, and $\gamma(p^{\mathcal{I}}(d)) = p^{\mathcal{J}}(\gamma(d))$. Hence, if γ is defined on $d \in \mathbf{D}^{\mathcal{I}}$, then γ is defined on every component of d and maps every component of d to a component of $\gamma(d)$.*

Proposition 3.2 *Let G be a feature graph. Then the components of G in $\mathcal{F}$ are exactly the maximal subgraphs of G. Furthermore, for every atom or variable s occurring in G there exists a unique component of G whose root is s, and all components of G can be obtained this way.*

Theorem 3.3 *The subsumption preorder of the feature graph algebra $\mathcal{F}$ is characterized as follows: $G \preceq^{\mathcal{F}} G'$ if and only if there exists a mapping θ from the variables and atoms occurring in G to the variables and atoms occurring in G' such that*

1. *θ maps the root of G to the root of G'*
2. *if a is an atom occurring in G, then $\theta a = a$*
3. *if xfs is an edge of G, then $(\theta x)f(\theta s)$ is an edge of G'.*

Proof. *1.* Let γ be a partial endomorphism of $\mathcal{F}$ such that $\gamma(G) = G'$. For every atom or variable s occurring in G we define θs to be the root of $\gamma(G_s)$. Since γ maps the components of G to components of G', we have $\gamma(G_s) = G'_{\theta s}$ for every variable or atom s occurring in G. If s is the root of G, then θs is the root of G' since $\gamma(G) = G'$. If a is an atom occurring in G, then $G'_{\theta a} = \gamma(G_a) = \gamma(a^{\mathcal{F}}) = a^{\mathcal{F}}$, and hence $\theta a = a$. If xfs is an edge of G, then

$$G'_{\theta s} = \gamma(G_s) = \gamma(f^{\mathcal{F}}(G_x)) = f^{\mathcal{F}}(\gamma(G_x)) = f^{\mathcal{F}}(G'_{\theta x}),$$

and hence $(\theta x)f(\theta s)$ is an edge of G'.

2. Let G and G' be feature graphs and θ be a mapping as required. Then we define $\gamma(G_s) = G'_{\theta s}$ for every variable or atom s occurring in G.

Then we have $\gamma(G) = \gamma(G_s) = G'_{\theta s} = G'$, where s is the root of G.

If γ is defined on $a^{\mathcal{F}}$, then $\gamma(a^{\mathcal{F}}) = \gamma(G_a) = G'_{\theta a} = G'_a = a^{\mathcal{F}}$.

Let G_x be a component of G such that $f^{\mathcal{F}}$ is defined on G_x. We have to show that $\gamma(f^{\mathcal{F}}(G_x)) = f^{\mathcal{F}}(\gamma(G_x))$. Since $f^{\mathcal{F}}$ is defined on G_x, we know that G contains an edge xfs such that $f^{\mathcal{F}}(G_x) = G_s$. Consequently, G' contains the edge $(\theta x)f(\theta s)$. Hence $\gamma(f^{\mathcal{F}}(G_x)) = \gamma(G_s) = G'_{\theta s} = f^{\mathcal{F}}(G'_{\theta x}) = f^{\mathcal{F}}(\gamma(G_x))$. □

Corollary 3.4 *The subsumption preorder $\preceq^{\mathcal{F}}$ of the feature graph algebra $\mathcal{F}$ linear-time decidable.*

4 Feature Constraints

The basic strategy of this paper is to accommodate feature descriptions as sublanguages of Predicate Logic with equality. In the previous section we have seen that the available nonlogical symbols are restricted to atoms and features and that the admissible interpretations, called feature algebras, must satisfy certain restrictions. In this section we will study the admissible formulas.

Since we don't have proper function symbols, a term in the sense of Predicate Logic is either a variable or an atom. (Recall that features are accommodated as binary predicate symbols.) As stated before, the letters s and t will always denote terms. As atomic formulas we have $f(s,t)$ and $s \doteq t$, where f is a feature. Atomic formulas of the form $f(s,t)$ will be written $fs \doteq t$ to suggest the functionality of features. From this two forms of atomic formulas we can build complex formulas using the usual connectives and quantifiers of Predicate Logic. For convenience, we will introduce additional syntactic forms, which, however, will not add any further expressivity.

The notion of a variable assignment is crucial for assigning meaning to formulas containing variables. Most presentations of Predicate Logic define a three-place relation $\mathcal{I}, \alpha \models \phi$ (called satisfaction) holding if the formula ϕ is satisfied by the interpretation $\mathcal{I}$ assuming the variable assignment α. For our purposes it is more convenient to introduce a function

$$\phi^{\mathcal{I}} \quad = \quad \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid \mathcal{I}, \alpha \models \phi\}$$

that maps a formula ϕ and an interpretation $\mathcal{I}$ to the set of all variable assignments α into $\mathcal{I}$ such that $\mathcal{I}$ satisfies ϕ under α. The assignments in $\phi^{\mathcal{I}}$ are called the solutions of ϕ in $\mathcal{I}$. From this perspective it is natural to see a formula as a constraint that restricts the values the variables occurring in it can take. For instance, the formula age x $\doteq$ y admits exactly those assignments that assign to x a value on which the feature age is defined and yields the value assigned to y.[5]

[5] The notion of constraint used here is worked out in a more general form in [13].

Let $\mathcal{I}$ be a feature algebra. An **$\mathcal{I}$-assignment** is a mapping from the set of all variables to the domain of $\mathcal{I}$. We use $\mathrm{ASS}[\mathcal{I}]$ to denote the set of all $\mathcal{I}$-assignments. Furthermore, if α is an $\mathcal{I}$-assignment and $d \in \mathbf{D}^{\mathcal{I}}$, we use $\alpha[x \leftarrow d]$ to denote the $\mathcal{I}$-assignment obtained from α by mapping x to d rather than to $\alpha(x)$.

Here are the **constraints** we are going to use:

$$
\begin{array}{rcll}
\phi, \psi & \longrightarrow & ps \doteq qt & \textit{feature equation} \\
 & & ps{\uparrow} & \textit{divergence (“p undefined on s”)} \\
 & & \phi \wedge \psi & \textit{conjunction} \\
 & & \neg\phi & \textit{negation} \\
 & & \phi \vee \psi & \textit{disjunction} \\
 & & \phi \rightarrow \psi & \textit{implication} \\
 & & \exists x(\phi) & \textit{existential quantification} \\
 & & \forall x(\phi) & \textit{universal quantification.}
\end{array}
$$

Recall that p and q always stand for paths, and that s and t always stand for variables or atoms. Since paths can be empty, feature equations subsume the atomic constraints $fs \doteq t$ and $s \doteq t$. We will write $ps \not\doteq qt$ and $s \not\doteq t$ as abbreviations for $\neg ps \doteq qt$ and $\neg s \doteq t$, respectively. The **free variables** of a constraint are defined as in Predicate Logic. A constraint is called **closed** if it has no free variables. In the following ϕ and ψ always stand for constraints as defined above.

Let $\mathcal{I}$ be a feature algebra. For every $\mathcal{I}$-assignment α we define $s^{\mathcal{I}}_{\alpha}$ as follows: $s^{\mathcal{I}}_{\alpha} = \alpha(x)$ if s is the variable x, and $s^{\mathcal{I}}_{\alpha} = a^{\mathcal{I}}$ if s is the atom a. With that we define the **solutions** of constraints in $\mathcal{I}$ as follows:

$$
\begin{array}{rcl}
(ps \doteq qt)^{\mathcal{I}} & := & \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid \exists d \in \mathbf{D}^{\mathcal{I}}\colon\ (s^{\mathcal{I}}_{\alpha}, d) \in p^{\mathcal{I}} \wedge (t^{\mathcal{I}}_{\alpha}, d) \in q^{\mathcal{I}}\} \\
(ps{\uparrow})^{\mathcal{I}} & := & \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid \forall d \in \mathbf{D}^{\mathcal{I}}\colon\ (s^{\mathcal{I}}_{\alpha}, d) \notin p^{\mathcal{I}}\} \\
(\phi \wedge \psi)^{\mathcal{I}} & := & \phi^{\mathcal{I}} \cap \psi^{\mathcal{I}} \\
(\neg\phi)^{\mathcal{I}} & := & \mathrm{ASS}[\mathcal{I}] - \phi^{\mathcal{I}} \\
(\phi \vee \psi)^{\mathcal{I}} & := & \phi^{\mathcal{I}} \cup \psi^{\mathcal{I}} \\
(\phi \rightarrow \psi)^{\mathcal{I}} & := & (\mathrm{ASS}[\mathcal{I}] - \phi^{\mathcal{I}}) \cup \psi^{\mathcal{I}} \\
(\exists x(\phi))^{\mathcal{I}} & := & \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid \exists d \in \mathbf{D}^{\mathcal{I}}\colon\ \alpha[x \leftarrow d] \in \phi^{\mathcal{I}}\} \\
(\forall x(\phi))^{\mathcal{I}} & := & \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid \forall d \in \mathbf{D}^{\mathcal{I}}\colon\ \alpha[x \leftarrow d] \in \phi^{\mathcal{I}}\}.
\end{array}
$$

We call a constraint **satisfiable** if there exists a feature algebra in which it has a solution, and we call two constraints ϕ and ψ **equivalent** (written $\phi \sim \psi$) if $\phi^{\mathcal{I}} = \psi^{\mathcal{I}}$ for every feature algebra $\mathcal{I}$. Furthermore, we say that a feature algebra $\mathcal{I}$ **satisfies** a constraint ϕ if $\phi^{\mathcal{I}} = \mathrm{ASS}[\mathcal{I}]$. Note that $\mathcal{I}$ satisfies ϕ if and only if $\neg\phi$ has no solution in $\mathcal{I}$.

We will call the logic given by the class of feature algebras and the constraints introduced above **Feature Logic**. Feature Logic is parameterized with respect to the atoms, features and variables we employ.

Several of the introduced constraint forms are redundant. In particular, one can eliminate in linear time divergences, implications, disjunctions and universal quantifications using the

following equivalences:

$$\begin{aligned} ps{\uparrow} &\sim \neg\exists x(ps \doteq x) \quad \text{if } x \neq s \\ \phi \rightarrow \psi &\sim \neg\phi \vee \psi \\ \phi \vee \psi &\sim \neg(\neg\phi \wedge \neg\psi) \\ \forall x(\phi) &\sim \neg\exists y(\neg\phi). \end{aligned}$$

The class of all feature algebras can be axiomatized. Let FA be a minimal set of constraints satisfying the following conditions:

1. if a and b are distinct atoms, then FA contains the constraint

$$a \not\doteq b$$

2. if f is a feature, then FA contains a constraint

$$fx \doteq y \wedge fx \doteq z \quad \rightarrow \quad y \doteq z$$

3. if f is a feature and a is an atom, then FA contains a constraint

$$\neg\exists x(fa \doteq x).$$

Note that FA is finite if there are only finitely many atoms and features.

Proposition 4.1 *The class of feature algebras is exactly the subclass of interpretations of Predicate Logic that satisfy every constraint of FA.*

As a consequence of this proposition we inherit from Predicate Logic sound and complete deduction calculi for Feature Logic.

Proposition 4.2 *The set of unsatisfiable feature constraints is recursively enumerable.*

Proof. A constraint ϕ is unsatisfiable if and only if $\neg\phi$ is valid, that is, is satisfied by every feature algebra. Since the valid formulas of Predicate Logic are recursively enumerable and the class of feature algebras is axiomatizable in Predicate Logic, we know that the unsatisfiable feature constraints are recursively enumerable. □

Our interest in the following is to determine the computational complexity of deciding the satisfiability of constraints. It will turn out that in general the satisfiability of constraints is undecidable. However, for quantifier-free constraints deciding satisfiability will be shown to be an NP-complete problem. Moreover, for constraints built from feature equations, divergences of the form $fx{\uparrow}$, negated equations of the form $s \not\doteq t$, conjunctions, and existential quantifications we will exhibit a quadratic-time algorithm for deciding satisfiability.

We will give a number of simplification rules that reduce constraints to equivalent ones having a simpler syntactic structure. To this purpose, we define four subclasses of constraints.

A constraint is in **existential prenix form** (EPF) if it has the form

$$\exists x_1 \cdots \exists x_n(\phi),$$

where $n \geq 0$ and ϕ (the **matrix**) is a quantifier-free constraint.

A constraint is **primitive** if it has one of the following four forms: $fs \doteq t$, $fs{\uparrow}$, $s \doteq t$, $s{\not\doteq}t$.

A constraint is **basic** if it can be built from primitive constraints by conjunctions and disjunctions.

A constraint is **quasi-basic** if it is in EPF and its matrix is a basic constraint.

Proposition 4.3 *For every constraint in EPF one can compute in linear time an equivalent quasi-basic constraint.*

The transformation algorithm verifying this proposition consists of three phases. The first phase eliminates implications and pushes down negations using the following equivalences:

$$\begin{array}{rcl}
\phi \to \psi & \sim & \neg\phi \vee \psi \\
\neg(\phi \wedge \psi) & \sim & \neg\phi \vee \neg\psi \\
\neg(\phi \vee \psi) & \sim & \neg\phi \wedge \neg\psi \\
\neg\neg\phi & \sim & \phi \\
\neg ps \doteq qt & \sim & ps{\uparrow} \vee qt{\uparrow} \vee \exists x \exists y(ps \doteq x \wedge qt \doteq y \wedge x{\not\doteq}y) \\
 & & \quad \text{if } x \neq y \text{ and } x \text{ and } y \text{ are different from } s \text{ and } t \\
\neg(ps{\uparrow}) & \sim & \exists x(ps \doteq x) \quad \text{if } x \text{ is different from } s.
\end{array}$$

The second phase eliminates all nonbasic paths:

$$\begin{array}{rcl}
ps \doteq qt & \sim & \exists x(ps \doteq x \wedge qt \doteq x) \quad \text{if } x \text{ is different from } s \text{ and } t \\
fps \doteq t & \sim & \exists x(ps \doteq x \wedge fx \doteq t) \quad \text{if } x \text{ is different from } s \text{ and } t \\
pfs{\uparrow} & \sim & fs{\uparrow} \vee \exists x(fs \doteq x \wedge px{\uparrow}) \quad \text{if } x \text{ is different from } s.
\end{array}$$

The third and last phase pushes existential quantifiers upwards (the constraint $[x/y]\phi$ is obtained from ϕ by replacing all free occurrences of the variable x with y):

$$\begin{array}{rcl}
\exists x(\phi) & \sim & \exists y([x/y]\phi) \quad \text{if } y \text{ doesn't occur in } \phi \\
\exists x(\phi) \wedge \psi & \sim & \exists x(\phi \wedge \psi) \quad \text{if } x \text{ doesn't occur in } \psi \\
\exists x(\phi) \vee \psi & \sim & \exists x(\phi \vee \psi) \quad \text{if } x \text{ doesn't occur in } \psi.
\end{array}$$

This leads to EPF since the preceding transformations don't introduce universal quantifications and don't introduce existential quantifications that are in the scope of a negation.

The transformation of constraints in EPF to quasi-basic form is the first step of a solution algorithm for constraints in EPF. For the further steps of the algorithm it is convenient to

get rid of the wrapping existential quantifications. This is possible by employing a finer notion of equivalence.

Let V be a set of variables. Two constraints ϕ and ψ are **V-equivalent** if the following two conditions are satisfied for every feature algebra $\mathcal{I}$:

1. if α is a solution of ϕ in $\mathcal{I}$, then there exists a solution β of ψ in $\mathcal{I}$ such that α and β agree on V
2. if α is a solution of ψ in $\mathcal{I}$, then there exists a solution β of ϕ in $\mathcal{I}$ such that α and β agree on V.

If ϕ and ψ are V-equivalent, then ϕ is satisfiable if and only if ψ is satisfiable.

Proposition 4.4 *Let $\phi = \exists x_1 \cdots \exists x_n(\psi)$ be a constraint and V be a set of variables such that $x_i \notin V$ for $i \in 1..n$. Then ϕ and ψ are V-equivalent.*

Proposition 4.5 *For every finite set of variables V and ever constraint in EPF one can compute in linear time a V-equivalent basic constraint.*

The first step in solving basic constraints is the elimination of disjunctions by transforming to a disjunctive normal form. To this purpose, we define a **feature clause** to be a finite, possibly empty set of primitive constraints representing their conjunction. Consequently, the **solutions** of a feature clause C in a feature algebra $\mathcal{I}$ are defined as:

$$C^{\mathcal{I}} \quad := \quad \bigcap_{\phi \in C} \phi^{\mathcal{I}}.$$

A feature clause is **satisfiable** if there is at least one feature algebra in which it has a solution. Two feature clauses C and D are **equivalent** if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every feature algebra $\mathcal{I}$. The letters C and D will always denote feature clauses.

The **disjuncts** of a basic constraint ϕ are the feature clauses defined as follows:

$$\begin{aligned} \mathbf{D}[\phi] &:= \{\{\phi\}\} \quad \text{if } \phi \text{ is a primitive constraint} \\ \mathbf{D}[\phi \wedge \psi] &:= \{C \cup D \mid C \in \mathbf{D}[\phi] \ \wedge \ D \in \mathbf{D}[\psi]\} \\ \mathbf{D}[\phi \vee \psi] &:= \mathbf{D}[\phi] \cup \mathbf{D}[\psi]. \end{aligned}$$

Clearly, a basic constraint has at least one disjunct and only finitely many. Note that the number of disjuncts of a basic constraint can be exponential with respect to its size.

Proposition 4.6 *Let $\mathcal{I}$ be a feature algebra. Then an $\mathcal{I}$-assignment is a solution of a basic constraint if and only if it is a solution of one of its disjuncts. Hence a basic constraint is satisfiable if and only if one of its disjuncts is satisfiable.*

In the next section we will give a quadratic-time solution algorithm for feature clauses. Since every disjunct of a basic constraint can be obtained in nondeterministic polynomial time, the transformations of this section will yield that the satisfiability of constraints in EPF is decidable in nondeterministic polynomial time.

Proposition 4.7 *Deciding the satisfiability of basic constraints in NP-hard, even if the constraint don't contain atoms or features.*

Proof. It is well-know that deciding the satisfiability of propositional formulas in conjunctive normal form (CNF) is an NP-complete problem. We show the claim by giving a linear-time translation from propositional formulas in CNF to basic constraints such that a propositional formula is satisfiable if and only if its translation is a satisfiable constraint. For this translation we assume without loss of generality that propositional variables are variables in the sense of feature logic (syntactically, of course). Furthermore, we fix a variable x_0 that is different from all propositional variables. Now we translate a propositional formula in CNF by replacing every positive literal x with $x \doteq x_0$ and every negative literal $\neg x$ with $x \not\doteq x_0$. □

5 Solving Feature Clauses

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

1. every constraint in C has one of the following forms: $fx \doteq s$, $fx{\uparrow}$, $x \doteq s$, $x \not\doteq s$
2. if $x \doteq s$ is in C, then x occurs exactly once in C
3. if $fx \doteq s$ and $fx \doteq t$ are in C, then $s = t$
4. if $fx{\uparrow}$ is in C, then C contains no constraint $fx \doteq s$
5. if $x \not\doteq s$ is in C, then $x \neq s$.

Let $\mathcal{I}$ be a feature algebra and ϕ be a constraint. An $\mathcal{I}$-assignment α is called a **principal solution** of ϕ in $\mathcal{I}$ if $\alpha \in \phi^{\mathcal{I}}$ and $\alpha(x) \preceq^{\mathcal{I}} \beta(x)$ for every $\beta \in \phi^{\mathcal{I}}$ and every variable x. We will show that every solved feature clause has a principal solution in the feature graph algebra $\mathcal{F}$.

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 s is a variable or an atom, then

$$\mathrm{FG}[s, C] := \begin{cases} (s, \emptyset) & \text{if } s \text{ is an atom} \\ \mathrm{FG}[t, C] & \text{if } s \doteq t \in C \\ (s,\ \{xft \mid fx \doteq t \in C \ \wedge\ s \to_C^* x\}) & \text{otherwise} \end{cases}$$

defines a feature graph.

Lemma 5.1 *If C is a solved feature clause, then $\alpha(x) := \mathrm{FG}[x, C]$ is a principal solution of C in $\mathcal{F}$.*

Proof. *1.* First we show that α is a solution of C in $\mathcal{F}$.

1.1. Let $fx \doteq s \in C$. Then x is the root of $\alpha(x)$ and xfs is an edge of $\alpha(x)$. Furthermore, it is easy to verify that $\mathrm{FG}[x,C]_s = \mathrm{FG}[s,C]$. Hence $f^{\mathcal{F}}(\alpha(x)) = f^{\mathcal{F}}(\mathrm{FG}[x,C]) = \mathrm{FG}[x,C]_s = \mathrm{FG}[s,C] = s^{\mathcal{F}}_{\alpha}$.

1.2. Let $fx \uparrow\in C$. Then C contains no constraint $fx \doteq s$ or $x \doteq s$. Hence $\alpha(x) = \mathrm{FG}[x,C] = (x,\emptyset)$ and consequently $f^{\mathcal{F}}$ is not defined on $\alpha(x)$.

1.3. Let $x \doteq s \in C$. Then $\alpha(x) = \mathrm{FG}[x,C] = \mathrm{FG}[s,C] = s^{\mathcal{F}}_{\alpha}$.

1.4. Let $x \not\doteq s \in C$. Then C contains no constraint $x \doteq t$ or $s \doteq t$. Hence x is the root of $\alpha(x)$ and s is the root of $s^{\mathcal{F}}_{\alpha}$. Since C is solved, we know that $x \neq s$. Hence $\alpha(x) \neq s^{\mathcal{F}}_{\alpha}$.

2. It remains to show that α is principal. Let $\beta \in C^{\mathcal{F}}$ and let x be a variable. We have to show that $\alpha(x) \preceq^{\mathcal{F}} \beta(x)$. To this purpose, let θ be the function that maps every atom to itself and every variable x to the root of $\beta(x)$. Because of Theorem 3.3 it suffices to show that θ maps the root of $\alpha(x)$ to the root of $\beta(x)$, and that $(\theta y)f(\theta s)$ is an edge of $\beta(x)$ if yfs is an edge of $\alpha(x)$.

2.1.1. Suppose the root of $\alpha(x) = \mathrm{FG}[x,C]$ is the atom a. Then C contains the constraint $x \doteq a$. Hence $\theta a = a$ must be the root of $\beta(x)$.

2.1.2. Suppose the root of $\alpha(x)$ is x. Then θx is the root of $\beta(x)$ by the definition of θ.

2.1.3. Suppose the root of $\alpha(x) = \mathrm{FG}[x,C]$ is some variable $y \neq x$. Then C contains the constraint $x \doteq y$ and hence $\beta(x) = \beta(y)$. Since θy is the root of βy by definition of θ, we know that θy is the root of $\beta(x)$.

2.2. Suppose yfs is an edge of $\alpha(x) = \mathrm{FG}[x,C]$. Then $fy \doteq s \in C$ and hence $f^{\mathcal{F}}(\beta(y)) = s^{\mathcal{F}}_{\beta}$. Since θy is the root of $\beta(y)$ and θs is the root of $s^{\mathcal{F}}_{\beta}$, we know that $(\theta y)f(\theta s)$ is an edge of $\beta(y)$.

Since yfs is an edge of $\alpha(x)$, we know either that $x \rightarrow^*_C y$ or that there exists $x \doteq z \in C$ such that $z \rightarrow^*_C y$. Hence there exists a path p such that $p^{\mathcal{F}}(\beta(x)) = \beta(y)$ since $\beta \in C^{\mathcal{F}}$. Thus $\beta(y)$ is a subgraph of $\beta(x)$ and hence $(\theta y)f(\theta s)$ is an edge of $\beta(x)$. □

Let C be a feature clause. Then we use

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

Our solution algorithm for feature clauses attempts to transform feature clauses to solved form using the following simplification rules:

1. $x \doteq s \;\&\; C \;\rightarrow\; x \doteq s \;\&\; [x/s]C$ if x occurs in C and $x \neq s$
2. $a \doteq x \;\&\; C \;\rightarrow\; x \doteq a \;\&\; C$
3. $fx \doteq s \;\&\; fx \doteq t \;\&\; C \;\rightarrow\; fx \doteq s \;\&\; s \doteq t \;\&\; C$

4. $s \doteq s \ \& \ C \ \rightarrow \ C$

5. $fa{\uparrow} \ \& \ C \ \rightarrow \ C$

6. $a{\not\doteq}x \ \& \ C \ \rightarrow \ x{\not\doteq}a \ \& \ C$

7. $a{\not\doteq}b \ \& \ C \ \rightarrow \ C \quad$ if $a \neq b$.

Proposition 5.2 *Let C be a feature clause. Then:*

1. *if D is obtained from C by a simplification rule, then D is a feature clause that is equivalent to C*

2. *there is no infinite chain of simplification 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 feature clauses such that, for every $i \geq 1$, C_{i+1} is obtained from C_i by a simplification rule. First note that every variable occurring in some C_i must also occur in C_1, that is, simplification 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 simplification rule decreases the number of isolated variables, and that the first simplification rule increases this number. Hence we can assume without loss of generality that the infinite sequence doesn't employ the first simplification rule. However, it is easy to see that the remaining simplification rules cannot support an infinite sequence. □

A feature clause is called **normal** if no simplification rule applies to it.

Proposition 5.3 *For every feature clause one can compute in quadratic time an equivalent normal feature clause.*

Proof. Let C be a clause. By the previous proposition we know that we can compute a normal feature clause D that is equivalent to C using the simplification rules. The simplification of C to D can be done in quadratic time by employing the simplification rules together with an efficient union-find method [1] for maintaining equivalence classes of variables and atoms. □

A feature clause is called **clash-free** if it satisfies the following conditions:

1. C contains no constraint of the form $fa \doteq s$ or $s{\not\doteq}s$

2. C contains no constraint of the form $a \doteq b$ such that $a \neq b$

3. if C contains a constraint $fx{\uparrow}$, then C contains no constraint $fx \doteq s$.

Proposition 5.4 *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.*

Let ϕ be a constraint, x be a variable, and $\mathcal{I}$ be a feature algebra. Then $d \in \mathbf{D}^{\mathcal{I}}$ is called a **solution** of x in ϕ and $\mathcal{I}$ if there exists a solution $\alpha \in \phi^{\mathcal{I}}$ such that $\alpha(x) = d$. Furthermore, $d \in \mathbf{D}^{\mathcal{I}}$ is called a **principal solution** of x in ϕ and $\mathcal{I}$ if d is a solution of x in ϕ and I and $d \preceq^{\mathcal{I}} e$ for every solution e of x in ϕ and $\mathcal{I}$.

Theorem 5.5 *Let C be a feature clause 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. *C has a principal solution in $\mathcal{F}$.*

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 a principal feature graph solution of x in C.

Proof. Follows from Propositions 5.3and 5.4 and Lemma 5.1. □

Theorem 5.6 *A constraint ϕ in EPF is satisfiable if and only if it has a solution in the feature graph algebra $\mathcal{F}$. Furthermore, deciding the satisfiability of constraints in EPF is an NP-complete problem. Finally, there is an exponential-time algorithm that, given a constraint ϕ in EPF and a variable x, returns finitely many solutions $G_1, \ldots, G_n$ $(n \geq 0)$ of x in ϕ and $\mathcal{F}$ such that for every solution G of x in ϕ and $\mathcal{F}$ there exists an $i \in 1..n$ such that $G_i \preceq^{\mathcal{F}} G$.*

Proof. Follows from Propositions 4.5 and 4.6, Theorem 5.5, and Proposition 4.7. □

Besides other things, the theorem says that for the satisfiability of constraints in EPF the feature graph algebra is canonical, that is, as far as satisfiability is concerned it suffices to consider the feature graph algebra. However, this does not hold for general constraints:

Proposition 5.7 *Let f and g be two distinct features. Then the constraint*

$$\forall x(fx \doteq x \rightarrow gx \doteq x)$$

is satisfiable but has no solution in the feature graph algebra $\mathcal{F}$.

Proof. It is clear that the given constraint has no solution in $\mathcal{F}$ (assign the feature graph $(x, \{xfx\})$ to x). To show that the constraint is satisfiable, we construct a feature algebra $\mathcal{I}$ satisfying the given constraint as follows:

$$\begin{array}{rcl} \mathbf{D}^{\mathcal{I}} & := & \{x\} \cup \{a \mid a \text{ is an atom}\} \\ a^{\mathcal{I}} & := & a \quad \text{for every atom } a \\ f^{\mathcal{I}} & := & \{(x,x)\} \quad \text{for every feature } f. \end{array}$$

□

A feature algebra is called **finite** if its domain is a finite set.

Corollary 5.8 *A constraint in EPF is satisfiable if and only if it has a solution in some finite feature algebra, provided there are only finitely many atoms.*

Proof. Let ψ be a satisfiable constraint in EPF whose matrix is ϕ. Then ϕ is satisfiable. By the preceding theorem we know that ϕ has a solution α in $\mathcal{F}$. Without loss of generality we can assume that ϕ contains at least one variable. Now we construct a finite feature algebra $\mathcal{I}$ as follows:

$$\begin{array}{rcl} \mathbf{D}^{\mathcal{I}} & := & \{a^{\mathcal{F}} \mid a \text{ is an atom}\} \\ & & \quad \cup \{G \mid x \text{ occurs in } \phi \text{ and } G \text{ is a component of } \alpha(x)\} \\ a^{\mathcal{I}} & := & a^{\mathcal{F}} \quad \text{for every atom } a \\ f^{\mathcal{I}} & := & f^{\mathcal{F}} \cap (\mathbf{D}^{\mathcal{I}} \times \mathbf{D}^{\mathcal{I}}) \quad \text{for every feature } f. \end{array}$$

From α we obtain a solution of ϕ in $\mathcal{I}$ by mapping all variables that don't occur in ϕ to arbitrary elements of $\mathbf{D}^{\mathcal{I}}$. Hence ψ has a solution in $\mathcal{I}$. □

6 Feature Terms

We now introduce a new form of expressions, called feature terms, that denote sets in feature algebras. Feature terms generalize Kasper and Rounds' feature descriptions [23, 39] and Aït-Kaci's ψ-terms [2, 3, 4].

Here is the abstract syntax of **feature terms**:

S, T	$\longrightarrow$	a	*atom*
		x	*variable*
		$p{:}\, S$	*selection*
		$p{\uparrow}$	*divergence*
		$p{\downarrow}q$	*agreement*
		$p{\uparrow}q$	*disagreement*
		$-$	*bottom*
		$\top$	*top*
		$S \sqcap T$	*intersection*
		$S \sqcup T$	*union*
		$\neg S$	*complement*
		$\exists x(S)$	*existential quantification.*

The **free variables** of a feature term are defined as one would expect. A feature term is called **closed** if it has no free variables. In the following S and T will always stand for feature terms.

Given a feature algebra $\mathcal{I}$ and an $\mathcal{I}$-assignment α, the **denotation** $S^{\mathcal{I}}_{\alpha}$ of a feature term S in $\mathcal{I}$ under α is a subset of $\mathbf{D}^{\mathcal{I}}$ defined inductively as follows:

$$\begin{aligned}
a^{\mathcal{I}}_{\alpha} &= \{a^{\mathcal{I}}\} \\
x^{\mathcal{I}}_{\alpha} &= \{\alpha(x)\} \\
(p\colon S)^{\mathcal{I}}_{\alpha} &= \{d \in \mathbf{D}^{\mathcal{I}} \mid \exists e \in S^{\mathcal{I}}_{\alpha}\colon\ (d,e) \in p^{\mathcal{I}}\} \\
(p{\uparrow})^{\mathcal{I}}_{\alpha} &= \{d \in \mathbf{D}^{\mathcal{I}} \mid \forall e \in \mathbf{D}^{\mathcal{I}}\colon\ (d,e) \notin p^{\mathcal{I}}\} \\
(p{\downarrow}q)^{\mathcal{I}}_{\alpha} &= \{d \in \mathbf{D}^{\mathcal{I}} \mid \exists e \in \mathbf{D}^{\mathcal{I}}\colon\ (d,e) \in p^{\mathcal{I}} \cap q^{\mathcal{I}}\} \\
(p{\uparrow}q)^{\mathcal{I}}_{\alpha} &= \{d \in \mathbf{D}^{\mathcal{I}} \mid \exists e, e' \in \mathbf{D}^{\mathcal{I}}\colon\ (d,e) \in p^{\mathcal{I}} \wedge (d,e') \in q^{\mathcal{I}} \wedge e \neq e'\} \\
-^{\mathcal{I}}_{\alpha} &= \emptyset \\
\top^{\mathcal{I}}_{\alpha} &= \mathbf{D}^{\mathcal{I}} \\
(S \sqcap T)^{\mathcal{I}}_{\alpha} &= S^{\mathcal{I}}_{\alpha} \cap T^{\mathcal{I}}_{\alpha} \\
(S \sqcup T)^{\mathcal{I}}_{\alpha} &= S^{\mathcal{I}}_{\alpha} \cup T^{\mathcal{I}}_{\alpha} \\
(\neg S)^{\mathcal{I}}_{\alpha} &= \mathbf{D}^{\mathcal{I}} - S^{\mathcal{I}}_{\alpha} \\
(\exists x(S))^{\mathcal{I}}_{\alpha} &= \bigcup_{d \in \mathbf{D}^{\mathcal{I}}} S^{\mathcal{I}}_{\alpha[x \leftarrow d]}.
\end{aligned}$$

To use feature terms in constraints, we introduce a new constraint form called membership. A **membership** takes the form $x\colon S$, where x is a variable and S is a feature term. The solutions of a membership in a feature algebra $\mathcal{I}$ are defined as follows:

$$(x\colon S)^{\mathcal{I}} = \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid \alpha(x) \in S^{\mathcal{I}}_{\alpha}\}.$$

Feature terms and memberships provide for an attractive syntax for the lexical rules of unification grammars. For instance, the constraint of the lexical rule $\mathsf{V} \longrightarrow \mathsf{sings}$ of the grammar in Section 2 can be written equivalently as the membership

$$\mathsf{V}\colon \left[\begin{array}{l} \mathsf{tense}\colon \mathsf{present} \\ \mathsf{pred}\colon \mathsf{verb}\colon \mathsf{sing} \\ \mathsf{subj}\colon \left[\begin{array}{l} \mathsf{num}\colon \mathsf{sg} \\ \mathsf{person}\colon \mathsf{3rd} \end{array}\right] \\ \mathsf{agent\ pred}{\downarrow}\mathsf{subj} \\ \mathsf{what\ pred}{\downarrow}\mathsf{obj} \end{array}\right]$$

or, using variables to express agreement,

$$\mathsf{V}\colon \exists \mathsf{x} \exists \mathsf{y} \left[\begin{array}{l} \mathsf{tense}\colon \mathsf{present} \\ \mathsf{pred}\colon \left[\begin{array}{l} \mathsf{verb}\colon \mathsf{sing} \\ \mathsf{agent}\colon \mathsf{x} \\ \mathsf{what}\colon \mathsf{y} \end{array}\right] \\ \mathsf{subj}\colon \left[\begin{array}{l} \mathsf{x} \\ \mathsf{num}\colon \mathsf{sg} \\ \mathsf{person}\colon \mathsf{3rd} \end{array}\right] \\ \mathsf{obj}\colon \mathsf{y} \end{array}\right].$$

The feature terms above are written in matrix notation, which can be traced back to Kay's functional unification grammar [24]. The feature terms given as the rows of a matrix are connected by intersections.

We call two feature terms S and T **equivalent** (written $S \sim T$) if $S^{\mathcal{I}}_{\alpha} = T^{\mathcal{I}}_{\alpha}$ for every feature algebra $\mathcal{I}$ and every $\mathcal{I}$-assignment α.

Most of the introduced feature term forms are redundant. Every feature term can be rewritten in linear time to an equivalent feature term containing only the forms

$$a, \quad x, \quad f{:}\,S, \quad S \sqcap T, \quad \neg S, \quad \exists x(S)$$

by using the following equivalences (ϵ is the empty path):

$$\begin{array}{rcl}
\epsilon{:}\,S & \sim & S \\
pf{:}\,S & \sim & f{:}\,(p{:}\,S) \\
p{\uparrow} & \sim & \neg(p{:}\,\top) \\
p{\downarrow}q & \sim & \exists x(p{:}\,x \sqcap q{:}\,x) \\
p{\uparrow}q & \sim & \exists x(p{:}\,x \sqcap q{:}\,\neg x) \\
- & \sim & x \sqcap \neg x \\
\top & \sim & \neg - \\
S \sqcup T & \sim & \neg(\neg S \sqcap \neg T).
\end{array}$$

Proposition 6.1 *For every feature term one can compute in linear time an equivalent feature term containing only the forms a, x, $f{:}\,S$, $S \sqcap T$, $\neg S$, and $\exists x(S)$.*

Furthermore, every constraint containing memberships can be rewritten in linear time into a membership-free constraint by using the following equivalences in addition to the previous ones:

$$\begin{array}{rcll}
x{:}\,a & \sim & x \doteq a & \\
x{:}\,y & \sim & x \doteq y & \\
x{:}\,(f{:}\,S) & \sim & \exists y(fx \doteq y \wedge y{:}\,S) & \text{if } y \neq x \text{ and } y \text{ does not occur in } S \\
x{:}\,S \sqcap T & \sim & x{:}\,S \wedge x{:}\,T & \\
x{:}\,\neg S & \sim & \neg(x{:}\,S) & \\
x{:}\,(\exists y(S)) & \sim & \exists y(x{:}\,S) & \text{if } y \neq x \\
\exists x(S) & \sim & \exists y([x/y]S) & \text{if } y \text{ does not occur in } S.
\end{array}$$

Proposition 6.2 *For every constraint one can compute in linear time an equivalent constraint not containing memberships.*

A feature term is **quantifier-free** if it contains no quantifications $\exists x(S)$. A feature term is **basic** if it is quantifier-free and contains only complements of the form $\neg a$ or $\neg x$. Every

quantifier-free feature term can be rewritten in linear time to an equivalent basic feature term by using the following equivalences:

$$\begin{array}{rcl}
\neg\, p{:}\, S & \sim & p{\uparrow} \sqcup p{:}\, \neg S \\
\neg\, p{\uparrow} & \sim & p{:}\, \top \\
\neg\, p{\downarrow}q & \sim & p{\uparrow} \sqcup q{\uparrow} \sqcup p{\uparrow}q \\
\neg\, p{\uparrow}q & \sim & p{\uparrow} \sqcup q{\uparrow} \sqcup p{\downarrow}q \\
\neg - & \sim & \top \\
\neg \top & \sim & - \\
\neg(S \sqcap T) & \sim & \neg S \sqcup \neg T \\
\neg(S \sqcup T) & \sim & \neg S \sqcap \neg T \\
\neg\neg S & \sim & S.
\end{array}$$

Proposition 6.3 *For every quantifier-free feature term one can compute in linear time an equivalent basic feature term.*

A constraint is in **EPFM** (existential prenix form with memberships) if it has the form $\exists x_1 \cdots \exists x_n(\phi)$, where $n \geq 0$ and ϕ is a quantifier-free constraint possibly containing quantifier-free memberships.

Proposition 6.4 *For every constraint in EPFM one can compute in linear time an equivalent quasi-basic constraint (not containing memberships).*

We show the claim by extending the algorithm verifying Proposition 4.3. The first phase of this algorithm eliminates implications and pushes down negations. We extend this phase by the equivalence

$$\neg(x{:}\, S) \quad \sim \quad x{:}\, \neg S$$

and the equivalences given above for transforming quantifier-free feature terms to basic feature terms.

After all negations and complements have been pushed down, all memberships are eliminated by rewriting with the following equivalences:

$$\begin{array}{rcl}
x{:}\, a & \sim & x \doteq a \\
x{:}\, y & \sim & x \doteq y \\
x{:}\, (p{:}\, S) & \sim & \exists y(px \doteq y \wedge y{:}\, S) \quad \text{if } y \neq x \text{ and } y \text{ does not occur in } S \\
x{:}\, p{\uparrow} & \sim & px{\uparrow} \\
x{:}\, (p{\downarrow}q) & \sim & px \doteq qx \\
x{:}\, (p{\uparrow}q) & \sim & \exists y \exists z(px \doteq y \wedge qx \doteq z \wedge y{\neq}z) \\
 & & \quad \text{if } x, y \text{ and } z \text{ are pairwise distinct} \\
x{:}\, - & \sim & x{\neq}x \\
x{:}\, \top & \sim & x \doteq x
\end{array}$$

$$
\begin{aligned}
x\colon S \sqcap T &\sim x\colon S \wedge x\colon T \\
x\colon S \sqcup T &\sim x\colon S \vee x\colon T \\
x\colon \neg y &\sim x \dot{\neq} y \\
x\colon \neg a &\sim x \dot{\neq} a.
\end{aligned}
$$

Now the remaining two phases of the algorithm verifying Proposition 4.3 lead to quasi-basic form.

A feature term S is called **coherent** if there exists a feature algebra $\mathcal{I}$ and an $\mathcal{I}$-assignment α such that $S^{\mathcal{I}}_{\alpha} \neq \emptyset$. A feature term is called **incoherent** if it is not coherent.

A feature term S is **included** in a feature term T (written $S \preceq T$) if $S^{\mathcal{I}}_{\alpha} \subseteq T^{\mathcal{I}}_{\alpha}$ for every interpretation $\mathcal{I}$ and every $\mathcal{I}$-assignment α.

Coherence, inclusion and equivalence of feature terms are linear-time reducible to each other:

$$
\begin{aligned}
S \text{ incoherent} &\iff S \preceq - \iff S \sim - \\
S \preceq T &\iff S \sqcap \neg T \text{ incoherent} \\
S \sim T &\iff S \preceq T \ \wedge \ T \preceq S.
\end{aligned}
$$

Furthermore, coherence of feature terms is linear-time reducible to satisfiability of memberships: If x is a variable not occurring in a feature term S, then

$$
S \text{ coherent} \iff x\colon S \text{ satisfiable.}
$$

Proposition 6.5 *Deciding incoherence, inclusion and equivalence of quantifier-free feature terms are Co-NP-complete problems.*

Proof. It suffices to show that deciding coherence of quantifier-free feature terms is an NP-complete problem. Because of Proposition 6.4 and Theorem 5.6 we know that the problem is in NP. The NP-hardness follows since propositional formulas in CNF can be regarded as feature terms such that satisfiability becomes coherence (conjunction is regarded as intersection, disjunction as union, negation as complement, and propositional variables are regarded as variables of Feature Logic). □

A feature algebra is **infinite** if its domain is an infinite set. The feature graph algebra $\mathcal{F}$ is infinite since there are infinitely many variables and $(x, \emptyset)$ and $(y, \emptyset)$ are distinct feature graphs if x and y are distinct variables. We will now show that one can compute for every feature term S a quantifier-free feature term T such that S and T are equivalent in every infinite feature algebra.

A feature term is **simple** if it is basic and contains no unions. A feature term is in **disjunctive normal form** (DNF) if it has the form $S_1 \sqcup \ldots \sqcup S_n$, where $S_1, \ldots, S_n$ are simple feature terms. A basic feature term can be rewritten into DNF by pushing up the occurring unions with the following equivalences:

$$
\begin{aligned}
S \sqcap (T \sqcup U) &\sim (S \sqcap T) \sqcup (S \sqcap U) \\
(S \sqcup T) \sqcap U &\sim (S \sqcap U) \sqcup (T \sqcap U) \\
p\colon (S \sqcup T) &\sim p\colon S \sqcup p\colon T.
\end{aligned}
$$

Together with Proposition 6.3 we have:

Proposition 6.6 *For every quantifier-free feature term one can compute an equivalent feature term in DNF.*

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

Lemma 6.7 *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 $\mathcal{V}(T) = \mathcal{V}(S) - \{x\}$ and $(\exists x(S))^{\mathcal{I}}_{\alpha} = T^{\mathcal{I}}_{\alpha}$ for every infinite feature algebra $\mathcal{I}$ and every $\mathcal{I}$-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:

$$\begin{array}{ll}
\Pi^+_x(S) := \emptyset \text{ if } x \notin \mathcal{V}(S) & \Pi^-_x(S) := \emptyset \text{ if } x \notin \mathcal{V}(S) \\
\Pi^+_x(p\colon S) := \{qp \mid q \in \Pi^+_x(S)\} & \Pi^-_x(p\colon S) := \{qp \mid q \in \Pi^-_x(S)\} \\
\Pi^+_x(S \sqcap T) := \Pi^+_x(S) \cup \Pi^+_x(T) & \Pi^-_x(S \sqcap T) := \Pi^-_x(S) \cup \Pi^-_x(T) \\
\Pi^+_x(x) := \{\epsilon\} & \Pi^-_x(x) := \emptyset \\
\Pi^+_x(\neg x) := \emptyset & \Pi^-_x(\neg x) := \{\epsilon\}.
\end{array}$$

Now let S be a simple feature term, x be a variable, $\mathcal{I}$ be an infinite interpretation, and α be an $\mathcal{I}$-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 $(\exists x(S))^{\mathcal{I}}_{\alpha} = U^{\mathcal{I}}_{\alpha}$ since $\mathcal{I}$ is infinite. To see this note that

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

for every set M having at least $n+1$ elements (M^n is the n-fold cartesian product of M).

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 $(\exists x(S))^{\mathcal{I}}_{\alpha} = T^{\mathcal{I}}_{\alpha}$. □

Theorem 6.8 *For every feature term S one can compute a quantifier-free feature term T such that $S^{\mathcal{I}}_{\alpha} = T^{\mathcal{I}}_{\alpha}$ for every infinite feature algebra $\mathcal{I}$ and every $\mathcal{I}$-assignment α.*

Proof. It suffices to show that we can eliminate an innermost quantification. Hence we can assume without loss of generality that $S = \exists x(U)$, where U is quantifier-free. By rewriting U to DNF (Proposition 6.6) we obtain simple feature terms $U_1, \ldots, U_n$ such that

$$S \;=\; \exists x(U) \;\sim\; \exists x(U_1 \sqcup \ldots \sqcup U_n) \;\sim\; \exists x(U_1) \sqcup \ldots \sqcup \exists x(U_n).$$

By using the transformation of the preceding lemma for every disjunct we obtain simple feature terms $V_1, \ldots, V_n$ such that

$$S^{\mathcal{I}}_{\alpha} = (V_1 \sqcup \ldots \sqcup V_n)^{\mathcal{I}}_{\alpha}$$

for every infinite feature algebra $\mathcal{I}$ and every $\mathcal{I}$-assignment α. □

Corollary 6.9 *For every closed feature term S one can compute a variable-free feature term T such that $S^{\mathcal{I}}_{\alpha} = T^{\mathcal{I}}_{\alpha}$ for every infinite feature algebra $\mathcal{I}$ and every $\mathcal{I}$-assignment α.*

Corollary 6.10 *It is decidable whether for a feature term S there exists an $\mathcal{F}$-assignment α such that $S^{\mathcal{F}}_{\alpha} \neq \emptyset$.*

Proof. Let S be a feature term. By the preceding theorem we know that we can compute a quantifier-free feature term T such that $S^{\mathcal{F}}_{\alpha} = T^{\mathcal{F}}_{\alpha}$ for every $\mathcal{F}$-assignment α. Now let x be a variable that doesn't occur in T. Then there exists an $\mathcal{F}$-assignment α such that $S^{\mathcal{F}}_{\alpha} \neq \emptyset$ if and only if $x{:}T$ has a solution in $\mathcal{F}$. By Proposition 6.4 and Theorem 5.6 we know that is decidable whether $x{:}T$ has a solution in $\mathcal{F}$. □

7 Sorts

In this section we extend our logic to include sorts. For our purposes, a sort is simply a symbol denoting a subset of the domain of a feature algebra. Equivalently, one can regard a sort as a unary predicate. Our sorts correspond to the *concepts* of terminological languages [28, 33, 34] and to the *templates* of the PATR-II system [46]. They are different from sorts in sorted logics in that we don't exploit sorts to impose a well-sortedness discipline on formulas.

From now on we assume an additional alphabet whose symbols are called **sorts**. Furthermore, we assume that the primitive feature terms $\top$ and $-$ are sorts. A **proper sort** is a sort different from $-$ and $\top$. The letters A and B will always denote sorts.

To accommodate sorts semantically, we assume that every feature algebra $\mathcal{I}$ interprets every sort A as a set $A^{\mathcal{I}} \subseteq \mathbf{D}^{\mathcal{I}}$, where $\top^{\mathcal{I}} = \mathbf{D}^{\mathcal{I}}$ and $-^{\mathcal{I}} = \emptyset$. On a partial homomorphism $\gamma{:}\mathcal{I} \to \mathcal{J}$ we impose the additional requirement that $\gamma(d) \in A^{\mathcal{J}}$ if γ is defined on d and $d \in A^{\mathcal{I}}$.

If there are proper sorts, the feature graph algebra $\mathcal{F}$ is no longer an admissible feature algebra since it lacks their interpretations.

We extent the set of constraints by allowing for **sort atoms** of the form As having the solutions

$$(As)^{\mathcal{I}} \quad = \quad \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid s^{\mathcal{I}}_{\alpha} \in A^{\mathcal{I}}\}.$$

Furthermore, we extend the set of feature terms by stipulating that every sort is a feature term, where

$$A^{\mathcal{I}}_{\alpha} \quad = \quad A^{\mathcal{I}}.$$

It is straightforward to extend our solution algorithm for constraints to sorts. All we have to do is to strengthen the definition of clash-freeness by excluding clashes of the form $As \wedge \neg As$. To see that normal, clash-free feature clauses possibly containing sort literals are satisfiable, suppose that C is such a clause. Let D be obtained from C by deleting

all sort literals. Then D has a solution α in some feature algebra $\mathcal{I}$ that doesn't interpret sorts. We extend $\mathcal{I}$ to a feature algebra $\mathcal{J}$ interpreting sorts by

$$A^{\mathcal{J}} \quad = \quad \{s_\alpha^{\mathcal{I}} \mid As \in C\}.$$

Now α is a solution of C in $\mathcal{J}$ and hence C is satisfiable.

From this argumentation it becomes clear that there are infinitely many extensions of the feature graph algebra $\mathcal{F}$ to sorts and that none of this extensions is canonical in the sense $\mathcal{F}$ is canonical for constraints not containing proper sorts. However, we still have that every satisfiable constraint has a solution in some extension of $\mathcal{F}$.

Sorts become interesting if we add the possibility to define them. For instance, we may write the equations (here and in the following sorts are written slanted to distinguish them from atoms)

$$\begin{aligned} \textit{present3rdsg} &\doteq \left[\begin{array}{l} \mathsf{tense{:}\,present} \\ \mathsf{subj{:}}\left[\begin{array}{l} \mathsf{num{:}\,sg} \\ \mathsf{person{:}\,3rd} \end{array}\right] \end{array}\right] \\ \textit{transitive} &\doteq \left[\begin{array}{l} \mathsf{agent\ pred{\downarrow}subj} \\ \mathsf{what\ pred{\downarrow}obj} \end{array}\right] \end{aligned}$$

and admit only those feature algebras that satisfy these equations (that is, interpret the sort at the left-hand side with the same set they interpret the closed feature term at the right-hand side). With these two sort equations the constraint of the lexical rule $\mathsf{V} \longrightarrow \mathsf{sings}$ of the grammar in Section 2 can be written as the membership

$$\mathsf{V}\colon \left[\begin{array}{l} \mathsf{pred{:}\,verb{:}\,sing} \\ \textit{transitive} \\ \textit{present3rdsg} \end{array}\right].$$

Such sort equations are in fact supported by the unification grammar formalism PATR-II [46], where defined sorts are called templates. Sort definitions are a handy device for expressing lexical generalizations, which is important for large lexica.

Sort equations are an essential ingredient of so-called terminological logics used for knowledge representations. Besides sorts and features these logic also support so-called roles, which are binary relations not required to be functional. For instance, one might have a role "child" relating persons to their children. Nebel and Smolka [34] survey terminological logics and discuss their relations to feature-based unification grammars.

Sort equations can also be expressed without feature terms since $A \doteq S$ is logically equivalent to

$$\forall x(Ax \quad \leftrightarrow \quad x\colon S),$$

where the membership at the right-hand side of the equivalence can be rewritten as a constraint not containing feature terms.

As long as a set $\mathcal{D}$ of sort equations is noncyclic, (that is, no sort is defined with reference to itself), it is decidable whether a constraint ϕ in EPF has a solution in at least one model

of $\mathcal{D}$. To do this, we iteratively replace every defined sort in ϕ by the feature term defining it. Let $\phi_{\mathcal{D}}$ be the feature term eventually obtained from ϕ this way. Then $\phi_{\mathcal{D}}$ contains no defined sort. Now we decide whether ϕ_D is satisfiable ignoring $\mathcal{D}$. If there exists no feature algebra in which ϕ_D has a solution, then there is certainly no feature algebra satisfying $\mathcal{D}$ in which ϕ_D has a solution. Otherwise, ϕ_D has a solution α in some feature algebra $\mathcal{I}$. By updating the interpretations of the defined sorts according to $\mathcal{D}$, we obtain a feature algebra $\mathcal{J}$ such that α is still a solution of ϕ_D since $\phi_{\mathcal{D}}$ contains no defined sorts. Since $\mathcal{J}$ satisfies by construction every equation of $\mathcal{D}$, α is also a solution of ϕ in $\mathcal{D}$.

In the next section we will prove that in the presence of cyclic sort equations it is in general undecidable whether quantifier-free constraints have a solution in at least one model of the equations.

A **sort system** is a partial function from proper sorts to closed feature terms. A feature algebra $\mathcal{I}$ is a **model** of a sort system σ if $A^{\mathcal{I}} = \sigma(A)^{\mathcal{I}}$ for every sort on which σ is defined. There are sort systems that don't have a model, for instance, $A \doteq \neg A$.

A feature term is **definite** if it is equivalent to a feature term

$$\exists x_1 \cdots \exists x_n(S),$$

where S is quantifier-free and no sort in S occurs in the scope of a complement. A sort system σ is **definite** if $\sigma(A)$ is a definite sort term for every sort A on which σ is defined.

We will show that for every definite sort system σ every feature algebra not interpreting sorts can be extended to a model of σ, and that there is a unique minimal such model. The proof consists just of a straightforward application of the fundamental result of the theory of definite relations [13].

The **base** of a feature algebra is the feature algebra obtained by forgetting all sort interpretations. The following defines a partial order on feature algebras:

$$\begin{aligned} \mathcal{I} \leq \mathcal{J} \quad :\Longleftrightarrow \quad & \mathcal{I} \text{ and } \mathcal{J} \text{ have the same base and} \\ & A^{\mathcal{I}} \subseteq A^{\mathcal{J}} \text{ for every sort } A. \end{aligned}$$

Theorem 7.1 *Let σ be a definite sort system. Then, for every feature algebra $\mathcal{I}$ without sort interpretations, there exists a unique least model of σ whose base is $\mathcal{I}$.*

Proof. Follows from a theorem in [13] since σ can be expressed equivalently as a definite set of equivalences $Ax \leftrightarrow \phi$ over feature logic without sorts. □

One consequence of this theorem is that a definite sort system uniquely defines least sort interpretations for every feature algebra without sorts. In particular, this is the case for the feature graph algebra $\mathcal{F}$. Aït-Kaci's *knowledge bases* [3] are definite sort systems whose models are restricted to extensions of the feature graph algebra $\mathcal{F}$.

Rounds and Manaster-Ramer [40] show that there is a definite sort systems σ not involving variables, complements or disagreements such that it is undecidable whether a sort denotes

the empty set in the least model of σ extending $\mathcal{F}$. Their result depends on the availability of feature terms with unions.

As Aït-Kaci does in his ψ-term calculus [3], one can assume that a lattice ordering of the sorts is given, where $-$ is the least and $\top$ is the greatest sort. In this case one admits only those feature algebra that interpret the infimum of two sorts A and B (their greatest lower bound in the lattice) as the intersection of the interpretations of A and B. The algorithms given in this paper for constraints without sorts can be easily extended to accommodate the sort lattice and the complexity results shown here remain unchanged. For an elaboration of Feature Logic with sort lattices see [47].

8 Two Undecidability Results

In this section we show that the set of satisfiable constraints of Feature Logic is not recursively enumerable. Moreover, we show that there are recursive sort equations such that it is undecidable whether a feature term denotes a nonempty set in at least one model of the equations. Both results are shown by coding the word problem of Thue systems whose undecidability is well-known.

We start by defining a class of Thue systems that is convenient for our purposes.

Let $\mathbf{S}$ be the set of all atoms and features, $\mathbf{S}^*$ be the set of all words over $\mathbf{S}$, and ϵ be the empty word. Note that the words containing only features are exactly the paths we use in feature constraints. A **Thue equation** is a set $\{p, q\}$ consisting of two distinct, nonempty paths. A **Thue system** is a finite set T of Thue equations. Every Thue system T defines a binary relation

$$u \leftrightarrow_T w \quad :\Longleftrightarrow \quad \exists\, w_1, w_2 \in \mathbf{S}^*\ \exists\, \{p, q\} \in T\colon \quad u = w_1 p w_2 \ \wedge\ w = w_1 q w_2$$

on $\mathbf{S}^*$. We use $\sim_T$ to denote the reflexive and transitive closure of $\leftrightarrow_T$ on $\mathbf{S}^*$. It is easy to see that $\sim_T$ is an equivalence relation on $\mathbf{S}^*$ satisfying

$$u \sim_T u' \wedge w \sim_T w' \quad \Rightarrow \quad uw \sim_T u'w'.$$

If T is clear from the context, we use $\overline{w}$ to denote the equivalence class of a word $w \in \mathbf{S}^*$ with respect to $\sim_T$. Since the paths in Thue equations are nonempty, we have $\overline{\epsilon} = \{\epsilon\}$ and $\overline{a} = \{a\}$ for every atom a.

Proposition 8.1 *Suppose there are at least two features. Then there exists a Thue system T such that it is undecidable whether $p \sim_T q$ for two paths p and q (the so-called word problem of T).*

Proof. It is well-known that there exists a Thue system whose word problem is undecidable (see, for instance, [12] for a proof). Such an undecidable Thue system can be transformed into an undecidable Thue system meeting our specialized format. □

Lemma 8.2 *Let f_1 and f_2 be two features, $p, q \in \{f_1, f_2\}^*$, and let T be a Thue system such that T contains no features other than f_1 and f_2. Furthermore, let ϕ_T and $\phi_{p,q}$ be the feature constraints*

$$\begin{aligned} \phi_T &= \forall x(f_1 x{\downarrow} \rightarrow (f_1 f_1 x{\downarrow} \wedge f_1 f_2 x{\downarrow} \wedge \bigwedge_{\{u,w\} \in T} ux \doteq wx)) \\ \phi_{p,q} &= \forall x(f_1 x{\downarrow} \rightarrow px \doteq qx), \end{aligned}$$

where $fx{\downarrow}$ abbreviates $\neg fx{\uparrow}$ (read: f defined on x). Then

$$p \sim_T q \iff \textit{every feature algebra satisfies } \phi_T \rightarrow \phi_{p,q}.$$

Proof. Since ϕ_T and $\phi_{p,q}$ are closed constraints, they are satisfied by a feature algebra $\mathcal{I}$ if and only if they have a solution in $\mathcal{I}$.

"$\Rightarrow$" Let $p \sim_T q$ and let $\mathcal{I}$ be a feature algebra that satisfies ϕ_T. We have to show that $\mathcal{I}$ satisfies $\phi_{p,q}$. Let $M := \mathbf{D}(f_1^{\mathcal{I}})$. Since $\mathcal{I}$ satisfies ϕ_T, we know that $f_1^{\mathcal{I}}$ and $f_2^{\mathcal{I}}$ are total functions from M to M, and that every equation $\{u, w\} \in T$ holds on M, that is, the functions $u^{\mathcal{I}}$ and $w^{\mathcal{I}}$ agree on M and are total on M (since $u, w \in \{f_1, f_2\}^*$). Since $p, q \in \{f_1, f_2\}^*$ and p can be obtained from q by finitely many applications of the equations in T, we hence know that the functions $p^{\mathcal{I}}$ and $q^{\mathcal{I}}$ agree on M. Hence $\mathcal{I}$ satisfies $\phi_{p,q}$.

"$\Leftarrow$" Suppose every feature algebra satisfies $\phi_T \rightarrow \phi_{p,q}$. We have to show that $p \sim_T q$. To this purpose we construct a feature algebra $\mathcal{I}$ as follows:

$$\begin{aligned} \mathbf{D}^{\mathcal{I}} &:= \mathbf{S}^*/\!\sim_T \text{ (the quotient of } \mathbf{S}^* \text{ with respect to } \sim_T) \\ a^{\mathcal{I}} &:= \overline{a} = \{a\} \quad \text{for every atom } a \\ C &:= \{\overline{a} \mid a \text{ is an atom}\} \\ (\overline{u}, \overline{w}) \in f^{\mathcal{I}} &:\iff \overline{u} \notin C \wedge \overline{w} = \overline{fu} \quad \text{for every feature } f. \end{aligned}$$

It is easy to see that $\mathcal{I}$ satisfies ϕ_T. Since $\mathcal{I}$ satisfies the implication $\phi_T \rightarrow \phi_{p,q}$ by assumption, we know that $\mathcal{I}$ satisfies $\phi_{p,q}$. Thus $p^{\mathcal{I}}$ and $q^{\mathcal{I}}$ agree on $\overline{\epsilon}$ (the equivalence class of the empty word) and hence $\overline{p} = p^{\mathcal{I}}(\overline{\epsilon}) = q^{\mathcal{I}}(\overline{\epsilon}) = \overline{q}$, which yields $p \sim_T q$. □

Theorem 8.3 *If there are at least two features, then the set of satisfiable feature constraints is not recursively enumerable.*

Proof. Let T be a Thue system as required by the preceding lemma whose word problem is undecidable (exists by Proposition 8.1). Since the pairs (p, q) such that $p \sim_T q$ are recursively enumerable, we know that the pairs (p, q) such that $p \not\sim_T q$ are not recursively enumerable. Since we know by the preceding lemma that

$$p \not\sim_T q \iff \neg(\phi_T \rightarrow \phi_{p,q}) \text{ is satisfiable}$$

provided $p, q \in \{f_1, f_2\}^*$, the satisfiable feature constraints cannot be recursively enumerable. □

A related problem is the *satisfiability problem for the feature graph algebra* $\mathcal{F}$: Is it decidable whether a constraint has a solution in $\mathcal{F}$? We conjecture that the satisfiability

problem for $\mathcal{F}$ is decidable. Evidence for this conjecture comes from recent results [7, 29, 30] showing that related problems for the ground term algebra and the algebra of rational trees are decidable.

Next we show that coherence of feature terms with respect to models of recursive sort equations is undecidable.

Lemma 8.4 *Let B be a sort, b and c be two distinct atoms, and f_1, f_2 and h be three pairwise distinct features. Furthermore, let $p, q \in \{f_1, f_2\}^*$, and let $T = \{\{p_i, q_i\}\}_{i=1}^n$ be a Thue system such that T contains no features other than f_1 and f_2. Then $p \sim_T q$ if and only if the feature term*

$$B \sqcap hp{:}\,b \sqcap hq{:}\,c$$

denotes the empty set in every feature algebra satisfying the sort equation

$$B \doteq f_1{:}\,B \sqcap f_2{:}\,B \sqcap p_1{\downarrow}q_1 \sqcap \ldots \sqcap p_n{\downarrow}q_n.$$

Proof. "$\Rightarrow$" Let $p \sim_T q$ and let $\mathcal{I}$ be a feature algebra that satisfies the given sort equation. Furthermore, let $d \in B^{\mathcal{I}}$. Since $b^{\mathcal{I}} \neq c^{\mathcal{I}}$, it suffices to show that $p^{\mathcal{I}}(d) = q^{\mathcal{I}}(d)$. This follows with a similar argumentation as in the proof of Lemma 8.2.

"$\Leftarrow$" Suppose the given feature term denotes the empty set in every feature algebra satisfying the given sort equation. We have to show that $p \sim_T q$. To this purpose we construct a feature algebra $\mathcal{I}$ as follows:

$$\begin{array}{rcl}
\mathbf{D}^{\mathcal{I}} & := & \mathbf{S}^*/{\sim_T} \text{ (the quotient of } \mathbf{S}^* \text{ with respect to } \sim_T) \\
a^{\mathcal{I}} & := & \overline{a} = \{a\} \quad \text{for every atom } a \\
A^{\mathcal{I}} & := & \mathbf{D}^{\mathcal{I}} - \{\overline{a} \mid a \text{ is a atom}\} \quad \text{for every proper sort } A \\
(\overline{u}, \overline{w}) \in f^{\mathcal{I}} & :\Longleftrightarrow & \overline{u} \in B^{\mathcal{I}} \wedge \overline{w} = \overline{fu} \quad \text{for every feature } f \neq h \\
h^{\mathcal{I}}(\overline{u}) & := & \begin{cases} \overline{b} & \text{if } \overline{u} = \overline{p} \\ \overline{c} & \text{if } \overline{u} \in B^{\mathcal{I}} - \{\overline{p}\} \\ \text{undefined} & \text{otherwise.} \end{cases}
\end{array}$$

It is easy to verify that $\mathcal{I}$ satisfies the given sort equation. Hence we know by our assumptions that the given feature term denotes the empty set in $\mathcal{I}$. Since $\overline{\epsilon} \in \mathcal{B}^{\mathcal{I}}$, we thus know that $h^{\mathcal{I}}(p^{\mathcal{I}}(\overline{\epsilon})) \neq \overline{b}$ or $h^{\mathcal{I}}(q^{\mathcal{I}}(\overline{\epsilon})) \neq \overline{c}$. Since $h^{\mathcal{I}}(p^{\mathcal{I}}(\overline{\epsilon})) = h^{\mathcal{I}}(\overline{p\epsilon}) = h^{\mathcal{I}}(\overline{p}) = \overline{b}$, we know $h^{\mathcal{I}}(q^{\mathcal{I}}(\overline{\epsilon})) = h^{\mathcal{I}}(\overline{q}) \neq \overline{c}$. Hence $\overline{q} = \overline{p}$, which yields $p \sim_T q$. □

Theorem 8.5 *Suppose there are at least three features, two atoms, and one sort. Then there exists a sort equation E of the form*

$$A \doteq f{:}\,A \sqcap g{:}\,A \sqcap p_1{\downarrow}q_1 \sqcap \ldots \sqcap p_n{\downarrow}q_n$$

such that the feature terms of the form

$$A \sqcap p{:}\,a \sqcap q{:}\,b$$

that denote a nonempty set in at least one model of E are not recursively enumerable.

Proof. Follows from the preceding lemma using a similar argument as in the proof of Theorem 8.3. □

Note that this undecidability results applies to variable-free feature terms without unions and complements. The undecidability is caused by the presence of agreements. By adapting techniques used in [33] one can show that for variable-free feature terms not containing agreements or disagreements coherence with respect to recursive sort equations is decidable.

9 History and Related Work

The first two feature-based unification grammar formalisms were Kay's Functional Unification Grammar (FUG) [24, 25, 26] and Bresnan and Kaplan's Lexical-Functional Grammar (LFG) [17]. LFG uses context-free phrase structure rules augmented with feature constraints interpreted in the feature graph algebra. FUG, which to my knowledge resisted full formalization so far, is based on recursive equations between feature terms. FUG doesn't have phrase structure rules but instead has special constraints for establishing word order. Part of FUG has been formalized by Manaster-Ramer and Rounds [40].

LFG and FUG are rather different in that LFG employs feature equations interpreted in the feature graph algebra while FUG relies solely of feature terms whose interpretation is left open. Correspondingly, the operational semantics of LFG was presented as constraint solving while the major operation of FUG was outlined as feature term unification. The exact relationship between feature equations and feature terms was first worked out in [47].

Aït-Kaci's ψ-term calculus [2, 3, 4] is the first published formalization of feature terms. Aït-Kaci's ψ-terms are feature terms without complements, agreements and explicit quantification and are required to obey a rigid normal form. He defines by syntactical means a so-called subsumption ordering, which corresponds exactly to our inclusion ordering applied after closing the terms under existential quantification. Aït-Kaci shows that his subsumption ordering yields a lattice on the quotient of the set of all ψ-terms under equivalence. Furthermore, he gives an algorithm, called ψ-term unification, for computing the infimum of two ψ-terms in this lattice. Unification of two ψ-terms S and T corresponds to solving the constraint

$$x{:}\,\exists S \wedge x{:}\,\exists T,$$

where $\exists S$ is obtained from S by quantifying all free variables existentially. Aït-Kaci also outlines the model-theoretic semantics for feature terms given in this paper, but he makes no attempt to show that his syntactic subsumption ordering and the semantic inclusion ordering coincide (which, in fact, they do). Aït-Kaci's early work has been inspired by work on semantic networks and, in particular, KL-ONE [6]. An important difference, however, between feature terms and the descriptions employed in KL-ONE is that KL-ONE mainly relies on many-valued (that is, nonfunctional) attributes called roles.

Incidentally, when Aït-Kaci published his thesis [2] in 1984, Brachman and Levesque [5] published the by now standard semantics of KL-ONE, which models KL-ONE descriptions as set-denoting expressions and defines subsumption as set inclusion in all interpretations.

In 1986, Kasper and Rounds [23, 39] presented the first logical account of feature terms. They consider variable-free feature terms without complements and disagreements and define a satisfaction relation between feature graphs and feature terms. A feature graph satisfies a feature term in Kasper and Rounds' logic if and only if in our logic the graph is an element of the term's denotation in the feature graph algebra $\mathcal{F}$. Their work was inspired by FUG, which contributed the notion of a feature term, and Shieber's work on PATR-II [46, 44], which contributed the notions of feature graph and agreement (often called path equations).

In 1987, Aït-Kaci and Smolka [48] showed how feature-based inheritance hierarchies can be captured as algebraic specifications using order-sorted equational logic. They realized that memberships can be equivalently expressed without feature terms by equational constraints and that unification of feature terms corresponds to constraint solving.

In his thesis [16] published in 1987, Johnson develops a so-called Attribute-Value Logic that has much in common with the feature logic presented here. Johnson considers only quantifier-free constraints and does not study feature terms. His logic is somewhat more general than ours since he doesn't model features as partial functions but instead has an explicit application function. Hence the variables of his logic also range over features. Johnson proves that deciding satisfiability in his logic is an NP-complete problem. Johnson also formalizes a grammar formalism based on his logic that bears much resemblance with LFG.

The present paper is an elaboration of previous work of the author [47], which resulted from an effort to bring together the work of Aït-Kaci, Kasper/Rounds, and Johnson.

The constraint solving algorithm given here requires transformation into disjunctive normal form, which will usually cause an exponential blow-up in size. Kasper [21, 20] and Dörre and Eisele [11] have proposed better unification algorithms for feature terms that try to avoid pushing up unions as much as possible. A new constraint solving method of Dörre and Eisele [9] introduces so-called distributed disjunctions and works on feature constraints rather than feature terms.

Kasper [22] investigates the use of feature terms with implications (an implication $S \to T$ is equivalent to $\neg S \sqcup T$) for modeling systems in systemic grammar. He outlines a unification method for these terms.

Moshier and Rounds [31] study a feature term logic that interprets negations intuitionistically. They prove that the satisfiability problem of this logic is PSPACE-complete. Dawar and Vijay-Shanker [8] investigate several possible interpretations of negation in feature terms using three-valued logic.

The linguistic problem of so-called long-distance dependencies doesn't have a satisfactory solution in unification grammars relying on the constraints discussed here. However, with so-called functional uncertainty constraints an elegant solution is possible [19]. A **functional uncertainty constraint** takes the form $f^*x \doteq y$ and has the solutions

$$(f^*x \doteq y)^{\mathcal{I}} \quad = \quad \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid \exists n \geq 0\colon\ (f^n)^{\mathcal{I}}(\alpha(x)) = \alpha(y)\},$$

where f^n is the path consisting of exactly n occurrences of the feature f. In general, it is open whether the satisfiability of conjunctions of feature equations and functional

uncertainty constraints is decidable. Kaplan and Maxwell [18] given a decision algorithm for conjunctions satisfying a certain acyclicity condition.

Another interesting extension are subsumption constraints making the subsumption preorder of feature algebras syntactically available. A **subsumption constraint** takes the form $x \sqsubseteq y$ and has the solutions

$$(x \sqsubseteq y)^{\mathcal{I}} \quad = \quad \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid \alpha(x) \preceq^{\mathcal{I}} \alpha(y)\}.$$

Shieber [45] discusses the usefulness of subsumption constraints for dealing with coordination problems. Dörre and Rounds [10] show that the satisfiability of conjunctions of feature equations and subsumption constraints is undecidable. This problem is closely related to the semi-unification problem for first-order terms, whose undecidability has been shown recently [27].

Yet another useful extension are roles or set-valued features. **Roles** are symbols that are interpreted as set-valued functions

$$r^{\mathcal{I}} \colon \mathbf{D}^{\mathcal{I}} \to \mathbf{2}^{\mathbf{D}^{\mathcal{I}}}.$$

Features can be seen as special roles that map every element of the domain either to the empty set or to a singleton. Syntactically, roles can be accommodated as follows:

$$(srt)^{\mathcal{I}} \quad = \quad \{\alpha \in \mathrm{ASS}[\mathcal{I}] \mid t^{\mathcal{I}}_{\alpha} \in r^{\mathcal{I}}(s^{\mathcal{I}}_{\alpha})\}.$$

Furthermore, the following two constructs provide for the use of roles in feature terms:

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

As in predicate logic, these role quantifiers are complementary, that is, $\forall r(S)$ is equivalent to $\neg\exists(\neg S)$. We call generalized feature terms with role quantifiers **concept descriptions**. Variable-free concept descriptions are employed in so-called terminological logics that developed from the knowledge representation language KL-ONE [6]. Nebel and Smolka [34] give a survey of terminological logics and discuss their relation to feature logics. Nebel's monograph [33] is a thorough exposition of terminological logics. Deciding coherence of variable-free concept descriptions built from sorts, intersections, complements and role quantifications is a PSPACE-complete problem [42]. Hollunder [14] has shown that the coherence of variable-free concept descriptions (built with the constructs introduced so far) is decidable. Schmidt-Schauß [41] has shown that the generalization of the agreement construct to roles results in undecidability of the inclusion relation.

Rounds [38] investigates a generalization of feature graphs that accommodates set-valued edges.

The integration of Prolog-like logic programming with feature constraint languages seems to be a promising line of research. Language proposals based on this idea are LOGIN [4] and CIL [32]. The theoretical foundations for this kind of languages are given by Feature Logic and the constraint logic programming model [15, 13].

Acknowledgements. The research reported in this paper was carried out when I was a Postdoc Fellow with IBM Deutschland at the IWBS in Stuttgart and has been funded by the EUREKA Project Protos (EU 56). I'm grateful to Hassan Aït-Kaci for arousing my interest in feature descriptions during my stay at MCC in July of 1986. When I joined the LILOG project of IBM Deutschland in Stuttgart in January 1988, 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 formalism [50] provided me with the challenge to capture some of it in logic. Many discussions with Jochen Dörre, Markus Höhfeld and Bill Rounds helped me in working out the ideas leading to this paper.

References

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

[2] 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.

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

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

[5] 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, TX, Aug. 1984.

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

[7] H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7:371–425, 1989.

[8] A. Dawar and K. Vijay-Shanker. A three-valued interpretation of negation in feature structure descriptions. In *Proceedings of the 27th Annual Meeting of the ACL, University of British Columbia*, pages 18–24, Vancouver, B.C., Canada, June 1989.

[9] J. Dörre and A. Eisele. Determining consistency of feature terms with distributed disjunctions. In D. Metzing, editor, *GWAI-89, 13th German Workshop on Artificial Intelligence*, pages 270–279. Informatik Fachberichte 216, Springer-Verlag, 1989.

[10] J. Dörre and W. C. Rounds. On subsumption and semi-unification in feature algebras. Iwbs report, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, 1989. To appear in Proc. of LICS'90.

[11] 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.

[12] H. Hermes. *Enumerability, Decidability, Computability*. Springer-Verlag, Berlin, Germany, 1965.

[13] 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. Accepted for the Journal of Logic Programming.

[14] B. Hollunder. Subsumption algorithms for some attributive concept descriptions languages. Master's thesis, FB Informatik, Universität Kaiserslautern, 6750 Kaiserslautern, W. Germany, 1989.

[15] 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, Germany, Jan. 1987.

[16] 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.

[17] 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. The MIT Press, Cambridge, MA, 1982.

[18] R. M. Kaplan and J. T. Maxwell III. An algorithm for functional uncertainty. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 297–302, Budapest, Hungary, 1989.

[19] R. M. Kaplan and J. T. Maxwell III. Constituent coordination in lexical-functional grammar. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 303–305, Budapest, Hungary, 1989.

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

[21] 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.

[22] R. T. Kasper. Conditional descriptions in functional unification grammar. In *Proceedings of the 26th Annual Meeting of the ACL, State University of New York at Buffalo*, pages 233–240, Buffalo, New York, 1988.

[23] 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.

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

[25] M. Kay. Unification grammar. Technical report, Xerox PARC, Palo Alto, CA, 1983.

[26] 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.

[27] A. Kfoury, J. Tiuryn, and P. Urzyczyn. The undecidability of the semi-unification problem. Technical report, Computer Science Department, Boston University, Boston, MA, October 1989.

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

[29] M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. Research report, IBM, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, U.S.A., 1988.

[30] M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science*, pages 348–457, Edinburgh, Scotland, July 1988.

[31] 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.

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

[33] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Germany, 1990.

[34] B. Nebel and G. Smolka. Representation and reasoning with attributive descriptions. In K. Bläsius, U.Hedtstück, and C.-R. Rollinger, editors, *Sorts and Types in Artificial Intelligence*, volume 418 of *Lectures Notes in Artificial Intelligence*, pages 112–139. Springer-Verlag, Berlin, Germany, 1990.

[35] 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. The MIT Press.

[36] F. C. Pereira and D. H. 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.

[37] 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.

[38] W. C. Rounds. Set values for unification-based grammar formalisms and logic programming. Report CSLI-88-129, Center for the Study of Language and Information, Stanford University, CA, 1988.

[39] 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.

[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, CA, 1987.

[41] 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, Canada, May 1989.

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

[43] 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, CA, 1984.

[44] 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.

[45] S. M. Shieber. Parsing and type inference for natural and computer languages. Technical Note 460, SRI International, Artificial Intelligence Center, Menlo Park, CA, Mar. 1989.

[46] S. M. Shieber, H. Uszkoreit, F. C. 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, CA, 1983.

[47] G. Smolka. A feature logic with subsorts. LILOG Report 33, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, May 1988. To appear in: J. Wedekind and C. Rohrer (eds.), Unification in Grammar; The MIT Press, 1991.

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

[49] H. Uszkoreit. Categorical Unification Grammar. In *Proceedings of the 11th International Conference on Computational Linguistics*, pages 187–194, Bonn, Germany, 1986.

[50] 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.