FEATURE LOGIC WITH WEAK SUBSUMPTION CONSTRAINTS

Jochen Dörre
IBM Deutschland GmbH
Science Center – IKBS
P.O. Box 80 08 80
D-7000 Stuttgart 80, Germany

ABSTRACT

In the general framework of a constraint-based grammar formalism often some sort of feature logic serves as the constraint language to describe linguistic objects. We investigate the extension of basic feature logic with subsumption (or matching) constraints, based on a weak notion of subsumption. This mechanism of one-way information flow is generally deemed to be necessary to give linguistically satisfactory descriptions of coordination phenomena in such formalisms. We show that the problem whether a set of constraints is satisfiable in this logic is *decidable* in polynomial time and give a *solution algorithm.*

1 Introduction

Many of the current constraint-based grammar formalisms, as e.g. FUG [Kay 79, Kay 85], LFG [Kaplan/Bresnan 82], HPSG [Pollard/Sag 87], PATR-II [Shieber et al. 83] and its derivates, model linguistic knowledge in recursive feature structures. Feature (or functional) equations, as in LFG, or feature terms, as in FUG or STUF [Bouma et al. 88], are used as constraints to describe declaratively what properties should be assigned to a linguistic entity. In the last few years, the study of the formal semantics and formal properties of logics involving such constraints has made substantial progress [Kasper/Rounds 86, Johnson 87, Smolka 88, Smolka 89], e.g., by making precise which sublanguages of predicate logic it corresponds to. This paves the way not only for reliable implementations of these formalisms, but also for extensions of the basic logic with a precisely defined meaning. The extension we present here, weak subsumption constraints, is a mechanism of one-way information flow, often proposed for a logical treatment of coordination in a feature-based unification grammar.[1] It can be informally described as a device, which enables us to require that one part of a (solution) feature structure has to be subsumed (be an instance of) another part.

Consider the following example of a coordination with "and", taken from [Shieber 89].

(1) Pat hired [$_{NP}$ a Republican] and [$_{NP}$ a banker].
(2) *Pat hired [$_{NP}$ a Republican] and [$_{AP}$ proud of it].

Clearly (2) is ungrammatical since the verb "hire" requires a noun phrase as object complement and this requirement has to be fulfilled by both coordinated complements. This subcategorization requirement is modeled in a unification-based grammar generally using equations which cause the features of a complement (or parts thereof encoding the type) to get unified with features encoding the requirements of the respective position in the subcategorization frame of the verb. Thus we could assume that for a coordination the type-encoding features of *each element* have to be "unified into" the respective position in the subcategorization frame. This entails that the coordinated elements are taken to be of one single type, which then can be viewed as the type of the whole coordination. This approach works fine for the verb "hire", but certain verbs, used very frequently, do not require this strict identity.

(3) Pat has become [$_{NP}$ a banker] and [$_{AP}$ very conservative].
(4) Pat is [$_{AP}$ healthy] and [$_{PP}$ of sound mind].

The verb "become" may have either noun-phrase or adjective-phrase complements, "to be" allows prepositional and verb phrases in addition, and these may appear intermixed in a coordination. In order to allow for such "polymorphic" type requirements, we want to

[1] Another application would be type inference in a grammar formalism (or programming language) that uses a type discipline with polymorphic types.

state, that (the types of) coordinated arguments each should be an instance of the respective requirement from the verb. Expressed in a general rule for (constituent) coordination, we want the structures of coordinated phrases to be instances of the structure of the coordination. Using subsumption constraints the rule basically looks like this:

$$E \longrightarrow C \text{ and } D$$
$$E \sqsubseteq C$$
$$E \sqsubseteq D$$

With an encoding of the types like the one proposed in HPSG we can model the subcategorization requirements for "to be" and "to become" as generalizations of all allowed types (cf. Fig. 1).

$$\texttt{NP}=\begin{bmatrix}\texttt{n:} & +\\ \texttt{v:} & -\\ \texttt{bar:} & 2\end{bmatrix} \quad \texttt{AP}=\begin{bmatrix}\texttt{n:} & +\\ \texttt{v:} & +\\ \texttt{bar:} & 2\end{bmatrix}$$
$$\texttt{VP}=\begin{bmatrix}\texttt{n:} & -\\ \texttt{v:} & +\\ \texttt{bar:} & 2\end{bmatrix} \quad \texttt{PP}=\begin{bmatrix}\texttt{n:} & -\\ \texttt{v:} & -\\ \texttt{bar:} & 2\end{bmatrix}$$

'to be' requires: $\begin{bmatrix}\texttt{bar:} & 2\end{bmatrix}$ 'to become' requires: $\begin{bmatrix}\texttt{n:} & +\\ \texttt{bar:} & 2\end{bmatrix}$

Figure 1: Encoding of syntactic type

A similar treatment of constituent coordination has been proposed in [Kaplan/Maxwell 88], where the coordinated elements are required to be in a set of feature structures and where the feature structure of the whole set is defined as the generalization (greatest lower bound w.r.t. subsumption) of its elements. This entails the requirement stated above, namely that the structure of the coordination subsumes those of its elements. In fact, it seems that especially in the context of set-valued feature structures (cf. [Rounds 88]) we need some method of inheritance of constraints, since if we want to state general combination rules which apply to the set-valued objects as well, we would like constraints imposed on them to affect also their members in a principled way.

Now, recently it turned out that a feature logic involving subsumption constraints, which are based on the generally adopted notion of subsumption for feature graphs is undecidable (cf. [Dörre/Rounds 90]). In the present paper we therefore investigate a weaker notion of subsumption, which we can roughly characterize as relaxing the constraint that an instance of a feature graph contains all of its path equivalencies. Observe, that path equivalencies play no role in the subcategorization requirements in our examples above.

2 Feature Algebras

In this section we define the basic structures which are possible interpretations of feature descriptions, the expressions of our feature logic. Instead of restricting ourselves to a specific interpretation, like in [Kasper/Rounds 86] where feature structures are defined as a special kind of finite automata, we employ an open-world semantics as in predicate logic. We adopt most of the basic definitions from [Smolka 89]. The mathematical structures which serve us as interpretations are called feature algebras.

We begin by assuming the pairwise disjoint sets of symbols L, A and V, called the sets of features (or labels), atoms (or constants) and variables, respectively. Generally we use the letters f, g, h for features, a, b, c for atoms, and x, y, z for variables. The letters s and t always denote variables or atoms. We assume that there are infinitely many variables.

A *feature algebra* $\mathcal{A}$ is a pair $(D^{\mathcal{A}}, \cdot^{\mathcal{A}})$ consisting of a nonempty set $D^{\mathcal{A}}$ (the domain of $\mathcal{A}$) and an interpretation $\cdot^{\mathcal{A}}$ defined on L and A such that

- $a^{\mathcal{A}} \in D^{\mathcal{A}}$ for $a \in A$. (atoms are constants)
- If $a \neq b$ then $a^{\mathcal{A}} \neq b^{\mathcal{A}}$. (unique name assumption)
- If f is a feature then $f^{\mathcal{A}}$ is a unary partial function on $D^{\mathcal{A}}$. (features are functional)
- No feature is defined on an atom.

Notation. We write function symbols on the right following the notation for record fields in computer languages, so that $f(d)$ is written df. If f is defined at d, we write $df\downarrow$, and otherwise $df\uparrow$. We use p, q, r to denote strings of features, called paths. The interpretation function $\cdot^{\mathcal{A}}$ is straightforwardly extended to paths: for the empty path ε, $\varepsilon^{\mathcal{A}}$ is the identity on $D^{\mathcal{A}}$; for a path $p = f_1 \ldots f_n$, $p^{\mathcal{A}}$ is the unary partial function which is the composition of the functions $f_1^{\mathcal{A}} \ldots f_n^{\mathcal{A}}$, where $f_1^{\mathcal{A}}$ is applied first.

A feature algebra of special interest is the **Feature Graph Algebra** $\mathcal{F}$, since it is canonical in the sense that whenever there exists a solution for a formula in basic feature logic in some feature algebra then there is also one in the Feature Graph Algebra. The same holds if we ex-

tend our logic to subsumption constraints (see [Dörre/Rounds 90]). A feature graph is a rooted and connected directed graph. The nodes are either variables or atoms, where atoms may appear only as terminal nodes. The edges are labeled with features and for every node no two outgoing edges may be labeled with the same feature.

We formalize feature graphs as pairs (s_0, E) where $s_0 \in V \cup A$ is the root and $E \subseteq V \times L \times (V \cup A)$ is a set of triples, the edges. The following conditions hold:

1. If $s_0 \in A$, then $E = \emptyset$.
2. If (x, f, s) and (x, f, t) are in E, then $s = t$.
3. If (x, f, s) is in E, then E contains edges leading from the root s_0 to the node x.

Let $G = (x_0, E)$ be a feature graph containing an edge (x_0, f, s). The *subgraph under* f of G (written G/f) is the maximal graph (s, E') such that $E' \subseteq E$.

Now it is clear how the Feature Graph Algebra $\mathcal{F}$ is to be defined. $D^{\mathcal{F}}$ is the set of all feature graphs. The interpretation of an atom $a^{\mathcal{F}}$ is the feature graph $(a, \emptyset)$, and for a feature f we let $Gf^{\mathcal{F}} = G/f$, if this is defined. It is easy to verify that $\mathcal{F}$ is a feature algebra.

Feature graphs are normally seen as data objects containing information. From this viewpoint there exists a natural preorder, called subsumption preorder, that orders feature graphs according to their informational content thereby abstracting away from variable names. We do not introduce subsumption on feature graphs here directly, but instead we define a subsumption order on feature algebras in general.

Let $\mathcal{A}$ and $\mathcal{B}$ be feature algebras. A **simulation** between $\mathcal{A}$ and $\mathcal{B}$ is a relation $\Delta \subseteq D^{\mathcal{A}} \times D^{\mathcal{B}}$ satisfying the following conditions:

1. if $(a^{\mathcal{A}}, d) \in \Delta$ then $d = a^{\mathcal{B}}$, for each atom a, and
2. for any $d \in D^{\mathcal{A}}, e \in D^{\mathcal{B}}$ and $f \in L$: if $df^{\mathcal{A}} \downarrow$ and $(d, e) \in \Delta$, then $ef^{\mathcal{B}} \downarrow$ and $(df^{\mathcal{A}}, ef^{\mathcal{B}}) \in \Delta$.

Notice that the union of two simulations and the transitive closure of a simulation are also simulations.

A **partial homomorphism** γ between $\mathcal{A}$ and $\mathcal{B}$ is a simulation between the two which is a partial function. If $\mathcal{A} = \mathcal{B}$ we also call γ a *partial endomorphism.*

Definition. Let $\mathcal{A}$ be a feature algebra. The **(strong) subsumption preorder** $\sqsubseteq^{\mathcal{A}}$ and the **weak subsumption preorder** $\sqsubseteq_{\approx}^{\mathcal{A}}$ of $\mathcal{A}$ are defined as follows:

- d *(strongly) subsumes* e (written $d \sqsubseteq^{\mathcal{A}} e$) iff there is an endomorphism γ such that $\gamma(d) = e$.
- d *weakly subsumes* e (written $d \sqsubseteq_{\approx}^{\mathcal{A}} e$) iff there is a simulation Δ such that $d \Delta e$.

It can be shown (see [Smolka 89]) that the subsumption preorder of the feature graph algebra coincides with the subsumption order usually defined on feature graphs, e.g. in [Kasper/Rounds 86].

Example: Consider the feature algebra depicted in Fig. 2, which consists of the elements $\{1, 2, 3, 4, 5, a, b\}$ where a and b shall be (the pictures of) atoms and f, g, i and j shall be features whose interpretations are as indicated.

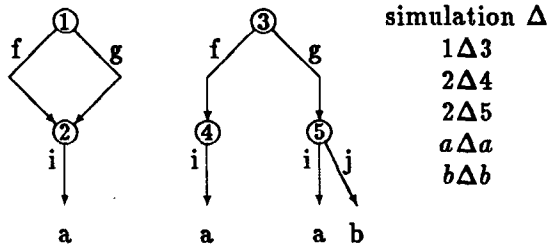

Figure 2: Example of Weak Subsumption

Now, element 1 does not strongly subsume 3, since for 3 it does not hold, that its f-value equals its g-value. However, the simulation Δ demonstrates that they stand in the weak subsumption relation: $1 \sqsubseteq_{\approx} 3$.

3 Constraints

To describe feature algebras we use a relational language similar to the language of feature descriptions in LFG or path equations in PATR-II. Our *syntax of constraints* shall allow for the forms

$$xp \doteq yq, \; xp \doteq a, \; xp \sqsubseteq_{\approx} yq$$

where p and q are paths (possibly empty), $a \in A$, and x and y are variables. A **feature clause** is a finite set of constraints of the above forms. As usual we interpret constraints with respect to a variable assignment, in order to make sure that variables are interpreted uniformly in the whole set. An **assignment** is a mapping α of variables to the elements of some feature alge-

bra. A constraint ϕ is satisfied in $\mathcal{A}$ under assignment α, written $(\mathcal{A},\alpha) \models \phi$, as follows:

$$(\mathcal{A},\alpha) \models xp \doteq yq \quad \text{iff} \quad \alpha(x)p^{\mathcal{A}} = \alpha(y)q^{\mathcal{A}}$$
$$(\mathcal{A},\alpha) \models xp \doteq a \quad \text{iff} \quad \alpha(x)p^{\mathcal{A}} = a^{\mathcal{A}}$$
$$(\mathcal{A},\alpha) \models xp \sqsubseteq yq \quad \text{iff} \quad \alpha(x)p^{\mathcal{A}} \sqsubseteq^{\mathcal{A}} \alpha(y)q^{\mathcal{A}}.$$

The **solutions** of a clause C in a feature algebra $\mathcal{A}$ are those assignments which satisfy each constraint in C. Two clauses C_1 and C_2 are **equivalent** iff they have the same set of solutions in every feature algebra $\mathcal{A}$.

The problem we want to consider is the following:

> *Given a clause C with symbols from V, L and A, does C have a solution in some feature algebra?*

We call this problem the weak semiunification problem in feature algebras.[2]

4 An Algorithm

4.1 Presolved Form

We give a solution algorithm for feature clauses based on normalization, i.e. the goal is to define a normal form which exhibits unsatisfiability and rewrite rules which transform each feature clause into normal form. The normal form we present here actually is only half the way to a solution, but we show below that with the use of a standard algorithm solutions can be generated from it.

First we introduce the restricted syntax of the normal form. Clauses containing only constraints of the following forms are called **simple**:

$$xf \doteq y,\ x \doteq s,\ x \sqsubseteq y$$

where s is either a variable or an atom. Each feature clause can be restated in linear time as an equisatisfiable simple feature clause whose solutions are extensions of the solutions of the original clause, through the introduction of auxiliary variables. This step is trivial.

A feature clause C is called **presolved** iff it is simple and satisfies the following conditions.

C1. If $x \doteq y$ is in C, then x occurs exactly once in C.
C2. If $xf \doteq y$ and $xf \doteq z$ are in C, then $y = z$.
C3. If $x \sqsubseteq y$ and $y \sqsubseteq z$ are in C, then $x \sqsubseteq z$ is in C (transitive closure).
C4. If $x \sqsubseteq y$ and $xf \doteq x'$ and $yf \doteq y'$ are in C, then $x' \sqsubseteq y'$ is in C (downward propagation closure).

In the first step our algorithm attempts to transform feature clauses to presolved form, thereby solving the equational part. In the simplification rules (cf. Fig. 3) we have adapted some of Smolka's rules for feature clauses including complements [Smolka 89]. In the rules $[x/s]C$ denotes the clause C where every occurrence of x has been replaced with s, and ϕ & C denotes the feature clause $\{\phi\} \cup C$ provided $\phi \notin C$.

Theorem 1 *Let C be a simple feature clause. Then*

1. *if C can be rewritten to D using one of the rules, then D is a simple feature clause equivalent to C,*
2. *for every non-normal simple feature clause one of the rewrite rules applies,*
3. *there is no infinite chain $C \to C_1 \to C_2 \to \ldots$*

Proof.[3] The first part can be verified straightforwardly by inspecting the rules. The same holds for the second part. To show the termination claim first observe that the application of the last two rules can safely be postponed until no one of the others can apply any more, since they only introduce subsumption constraints, which cannot feed the other rules. Now, call a variable x isolated in a clause C, if C contains an equation $x \doteq y$ and x occurs exactly once in C. The first rule strictly increases the number of isolated variables and no rule ever decreases it. Application of the second and third rule decrease the number of equational constraints or the number of features appearing in C, which no other rule increase. Finally, the last two rules strictly increase the number of subsumption constraints for a constant set of variables. Hence, no infinite chain of rewriting steps may be produced. □

We will show now, that the presolved form can be seen as a nondeterministic finite automaton

[2] The analogous problem for (strong) subsumption constraints is undecidable, even if we restrict ourselves to finite feature algebras. Actually, this problem could be shown to be equivalent to the semiunification problem for rational trees, i.e. first-order terms which may contain cycles. The interested reader is referred to [Dörre/Rounds 90].

[3] Part of this proof has been directly adapted from [Smolka 89].

$$x \doteq y \;\&\; C \;\rightarrow\; x \doteq y \;\&\; [x/y]C, \text{ if } x \text{ occurs in } C \text{ and } x \neq y \quad (1)$$

$$x \doteq x \;\&\; C \;\rightarrow\; C \quad (2)$$

$$xf \doteq y \;\&\; xf \doteq z \;\&\; C \;\rightarrow\; xf \doteq y \;\&\; y \doteq z \;\&\; C \quad (3)$$

$$x \sqsubseteq\!\!\!\sim y \;\&\; y \sqsubseteq\!\!\!\sim z \;\&\; C \;\rightarrow\; x \sqsubseteq\!\!\!\sim y \;\&\; y \sqsubseteq\!\!\!\sim z \;\&\; x \sqsubseteq\!\!\!\sim z \;\&\; C, \text{ if } x \sqsubseteq\!\!\!\sim z \notin C \quad (4)$$

$$x \sqsubseteq\!\!\!\sim y \;\&\; xf \doteq x' \;\&\; yf \doteq y' \;\&\; C \;\rightarrow\; x \sqsubseteq\!\!\!\sim y \;\&\; xf \doteq x' \;\&\; yf \doteq y' \;\&\; x' \sqsubseteq\!\!\!\sim y' \;\&\; C \quad (5)$$
$$\text{if } x' \sqsubseteq\!\!\!\sim y' \notin C$$

Figure 3: Rewriting to presolved form

with ε-moves and that we can read off solutions from its deterministic equivalent, if that is of a special, trivially verifiable, form, called clash-free.

4.2 The Transition Relation δ_C of a Presolved Clause C

The intuition behind this construction is, that subsumption constraints basically enforce that information about one variable (and the space reachable from it) has to be inherited by (copied to) another variable. For example the constraints $x \sqsubseteq\!\!\!\sim y$ and $xp \doteq a$ entail that also $yp \doteq a$ has to hold.[4] Now, if we have a constraint $x \sqsubseteq\!\!\!\sim y$, we could think of actually copying the information found under x to y, e.g. $xf \doteq x'$ would be copied to $yf \doteq y'$, where y' is a new variable, and x' would be linked to y' by $x' \sqsubseteq\!\!\!\sim y'$. However, this treatment is hard to control in the presence of cycles, which always can occur. Instead of actually copying we also can regard a constraint $x \sqsubseteq\!\!\!\sim y$ as a pointer from y back to x leading us to the information which is needed to construct the local solution of y. To extend this view we regard the whole presolved clause C as a finite automaton: take variables and atoms as nodes, a feature constraint as an arc labeled with the feature, constraints $x \doteq s$ and $y \sqsubseteq\!\!\!\sim x$ as ε-moves from x to s or y. We can show then that C is unsatisfiable iff there is some x from which we reach atom a via path p such that we can also reach $b(\neq a)$ via p or there is a path starting from x whose proper prefix is p.

Formally, let NFA $\mathcal{N}_C$ of presolved clause C be defined as follows. Its states are the variables occurring in C (V_C) plus the atoms plus the states q_F and the initial state q_0. The set of final states is $V_C \cup \{q_F\}$. The alphabet of $\mathcal{N}_C$ is $V_C \cup L \cup A \cup \{\varepsilon\}$.[5]

The transition relation is defined as follows:[6]

$$\begin{aligned}
\delta_C \;:=\; & \{(q_0, x, x) \mid x \in V_C\} \\
\cup\; & \{(a, a, q_F) \mid a \in A\} \\
\cup\; & \{(y, \varepsilon, x) \mid x \sqsubseteq\!\!\!\sim y \in C\} \\
\cup\; & \{(x, f, y) \mid xf \doteq y \in C\} \\
\cup\; & \{(x, \varepsilon, s) \mid x \doteq s \in C\}
\end{aligned}$$

As usual, let $\hat{\delta}_C$ be the extension of δ_C to paths. Notice that $xpa \in L(\mathcal{N}_C)$ iff $(x, p, a) \in \hat{\delta}_C$.

The language accepted by this automaton contains strings of the forms xp or xpa, where a string xp indicates that in a solution α the object $\alpha(x)p^{\mathcal{A}}$ should be defined and xpa tells us further that this object should be $a^{\mathcal{A}}$.

A set of strings of $(V \times L^*) \cup (V \times L^* \times A)$ is called **clash-free** iff it does not contain a string xpa together with xpb (where $a \neq b$) or together with xpf. It is clear that the property of a regular language L of being clash-free with respect to L and A can be read off immediately from a DFA $\mathcal{D}$ for it: if $\mathcal{D}$ contains a state q with $\delta(q, a) \in F$ and either $\delta(q, b) \in F$ (where $a \neq b$) or $\delta(q, f) \in F$, then it is not clash-free, otherwise it is.

We now present our central theorem.

Theorem 2 *Let C_0 be a feature clause, C its presolved form and $\mathcal{N}_C$ the NFA as constructed*

[4] From this point of view the difference between weak and strong subsumption can be captured in the type of information they enforce to be inherited. Strong subsumption requires path equivalences to be inherited ($x \sqsubseteq y$ and $xp \doteq xq$ implies $yp \doteq yq$), whereas weak subsumption does not.

[5] If L or A are infinite we restrict ourselves to the sets of symbols actually occurring in C.

[6] Notice that if $x \doteq s \in C$, then either s is an atom or x occurs only once. Thus it is pointless to have an arc from s to x, since we either have already the maximum of information for s or x will not provide any new arcs.

above. Then the following conditions are equivalent:

1. *$L(\mathcal{N}_C)$ is clash-free*
2. *There exists a* finite *feature algebra $\mathcal{A}$ and an assignment α such that $(\mathcal{A}, \alpha) \models C_0$, provided the set of atoms is finite.*
3. *There exists a feature algebra $\mathcal{A}$ and an assignment α such that $(\mathcal{A}, \alpha) \models C_0$.*

Proof. see Appendix A.

Now the algorithm consists of the following simple or well-understood steps:

1: (a) Solve the equational constraints of C, which can be done using standard unification methods, exemplified by rules 1) to 3).
   (b) Make the set of weak subsumption constraints transitively and "downward" closed (rules 4) and 5)).
2: The result interpreted as an NFA is made deterministic using standard methods and tested of being clash-free.

4.3 Determining Clash-Freeness Directly

For the purpose of proving the algorithm correct it was easiest to assume that clash-freeness is determined after transforming the NFA of the presolved form into a deterministic automaton. However, this translation step has a time complexity which is exponential with the number of states in the worst case. In this section[A we consider a technique to determine clash-freeness directly from the NFA representation of the presolved form in polynomial time. We do not go into implementational details, though. Instead we are concerned to describe the different steps more from a logical point of view. It can be assumed that there is still room left for optimizations which improve efficiency.

In a first step we eliminate all the ε-transitions from the NFA $\mathcal{N}_C$. We will call the result still $\mathcal{N}_C$. For every pair of a variable node x and an atom node a let $\mathcal{N}_C[x, a]$ be the (sub-)automaton of all states of $\mathcal{N}_C$ reachable from x, but with the atom a being the only final state. Thus, $\mathcal{N}_C[x, a]$ accepts exactly the language of all strings p for which $xpa \in L(\mathcal{N}_C)$. Likewise, let $\mathcal{N}_C[x, \bar{a}]$ be the (sub-)automaton of all states of $\mathcal{N}_C$ reachable from x, but where every atom node besides a is in the set of final states as well as every node with an outgoing feature arc. The set accepted by this machine contains every string p such that $xpb \in L(\mathcal{N}_C), (b \neq a)$ or $xpf \in L(\mathcal{N}_C)$. If and only if the intersection of these two machines is empty for every x and a, $L(\mathcal{N}_C)$ is clash-free.

4.4 Complexity

Let us now examine the complexity of the different steps of the algorithm.

We know that Part 1a) can be done (using the efficient union/find technique to maintain equivalence classes of variables and vectors of features for each representative) in nearly linear time, the result being smaller or of equal size than C_0. Part 1b) may blow up the clause to a size at most quadratic with the number of different variables n, since we cannot have more subsumption constraints than this. For every new subsumption constraint, trying to apply rule 4) might involve at most $2n$ membership test to check whether we are actually adding a new constraint, whereas for rule 5) this number only depends on the size of L. Hence, we stay within cubic time until here.

Determining whether the presolved form is clash-free from the NFA representation is done in three steps. The ε-free representation of $\mathcal{N}_C$ does not increase the number of states. If n, a and l are the numbers of variables, atoms and features resp. in the initial clause, then the number of edges is in any case smaller than $(n + a)^2 \cdot l$, since there are only $n + a$ states. This computation can be performed in time of an order less than $o((n + a)^3)$.

Second, we have to build the intersections for $\mathcal{N}_C[x, a]$ and $\mathcal{N}_C[x, \bar{a}]$ for every x and a. Intersection of two NFAs is done by building a cross-product machine, requiring maximally $o((n + a)^4 \cdot l)$ time and space.[7] The test for emptiness of these intersection machines is again trivial and can be performed in constant time.

Hence, we estimate a total time and space complexity of order $n \cdot a \cdot (n + a)^4 \cdot l$.

[7] This is an estimate for the number of edges, since the number of states is below $(n + a)^2$. As usual, we assume appropriate data structures where we can neglect the order of access times. Probably the space (and time) complexity can be reduced further, since we actually do not need the representations of the intersection machines besides for testing, whether they can accept anything.

5 Conclusion

We proposed an extension to the basic feature logic of variables, features, atoms, and equational constraints. This extension provides a means for one-way information passing. We have given a simple, but nevertheless completely formal semantics for the logic and have shown that the satisfiability (or unification) problem in the logic involving weak subsumption constraints is decidable in polynomial time. Furthermore, the first part of the algorithm is a surprisingly simple extension of a standard unification algorithm for feature logic. We have formulated the second part of the problem as a simple property of the regular language which the outcome of the first part defines. Hence, we could make use of standard techniques from automata theory to solve this part of the problem. The algorithm has been proved to be correct, complete, and guaranteed to terminate. There are no problems with cycles or with infinite chains of subsumption relations as generated by a constraint like $x \sqsubseteq xf$.[8]

The basic algorithmic requirements to solve the problem being understood, the challenge now is to find ways how solutions can be found in a more incremental way, if we already have solutions for subsets of a clause. To achieve this we plan to amalgamate more closely the two parts algorithms, for instance, through implementing the check for clash-freeness also with the help of (a new form of) constraints. It would be interesting also from a theoretical point of view to find out how much of the complexity of the second part is really necessary.

Acknowledgment

I am indebted to Bill Rounds for reading a first draft of this paper and pointing out to me a way to test clash-freeness in polynomial time. Of course, any remaining errors are those of the author. I would also like to thank Gert Smolka for giving valuable comments on the first draft.

References

[Bouma et al. 88] Gosse Bouma, Esther König and Hans Uszkoreit. A flexible graph-unification formalism and its application to natural-language processing. In: *IBM Journal of Research and Development*, 1988.

[Dörre/Rounds 90] Jochen Dörre and William C. Rounds. *On Subsumption and Semiunification in Feature Algebras.* In *Proceedings of the 5th Annual Symposium on Logic in Computer Science*, pages 300–310, Philadelphia, PA., 1990. Also appears in: Journal of Symbolic Computation.

[Johnson 87] Mark Johnson. *Attribute-Value Logic and the Theory of Grammar. CSLI Lecture Notes 16*, CSLI, Stanford University, 1987.

[Kaplan/Bresnan 82] Ronald M. Kaplan and Joan Bresnan. Lexical Functional Grammar: A Formal System for Grammatical Representation. In: J. Bresnan (ed.), *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts, 1982.

[Kaplan/Maxwell 88] Ronald M. Kaplan and John T. Maxwell III. Constituent Coordination in Lexical-Functional Grammar. In: *Proc. of COLING'88*, pp.303-305, Budapest, Hungary, 1988.

[Kasper/Rounds 86] Robert T. Kasper and William C. Rounds. A Logical Semantics for Feature Structures. In: *Proceedings of the 24th Annual Meeting of the ACL.* Columbia University, New York, NY, 1986.

[Kay 79] Martin Kay. Functional Grammar. In: C. Chiarello et al. (eds.) *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society.* 1979.

[Kay 85] Martin Kay. Parsing in Functional Unification Grammar. In: D. Dowty, L. Karttunen, and A. Zwicky (eds.) *Natural Language Parsing*, Cambridge, England, 1985

[Pollard/Sag 87] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics, Vol. 1. CSLI Lecture Notes 13*, CSLI, Stanford University, 1987.

[Rounds 88] William C. Rounds. *Set Values for Unification-Based Grammar Formalisms and Logic Programming.* CSLI-Report 88-129, CSLI, Stanford University, 1988.

[Shieber et al. 83] Stuart M. Shieber, Hans Uszkoreit, Fernando C.N. Pereira, J.J. Robinson, M. Tyson. The formalism and implementation of PATR-II. In: J. Bresnan (ed.), *Research on Interactive Acquisition and Use of Knowledge*, SRI International, Menlo Park, CA, 1983.

[Shieber 89] Stuart M. Shieber. Parsing and Type Inference for Natural and Computer Languages. Technical Note 460, SRI International, Menlo Park, CA, March 1989.

[Smolka 88] Gert Smolka. *A Feature Logic with Subsorts.* LILOG-Report 33, IWBS, IBM Deutschland, W. Germany, May 1988. To appear in the Journal of Automated Reasoning.

[Smolka 89] Gert Smolka. *Feature Constraint Logics for Unification Grammars.* IWBS Report 93, IWBS, IBM Deutschland, W. Germany, Nov. 1989. To appear in the Proceedings of the Workshop on Unification Formalisms—Syntax, Semantics and Implementation, Titisee, The MIT Press, 1990.

[8] See [Shieber 89] for a discussion of this problem.

Appendix A: Proof of Theorem 2

From Theorem 1 we know that C is equivalent to C_0, i.e. it suffices to show the theorem for the existence of solutions of C in 2) and 3). Since 2) $\Rightarrow$ 3) is obvious, it remains to show 1) $\Rightarrow$ 2) and 3) $\Rightarrow$ 1).

1) $\Rightarrow$ 2): We construct a finite model $(\mathcal{A}, \alpha)$ whose domain contains partial functions from paths to atoms ($D^{\mathcal{A}} \subset L^* \rightarrow A$). Interpretation and variable assignment are given as follows:

- $a^{\mathcal{A}} = \{(\epsilon, a)\}$ for every atom a (the function mapping only the empty string to a)
- for every $f \in L, \chi \in L^* \rightarrow A$: $\chi f^{\mathcal{A}} = \{(p,a) \mid (fp,a) \in \chi\}$
- $\alpha(x) = \{(p,a) \mid xpa \in L(\mathcal{N}_C)\}$, which is a partial function, due to 1).

Now let the elements of the domain be, besides interpretations of atoms, just those objects (partial functions) which can be reached by application of some features to some $\alpha(x)$.

- $D^{\mathcal{A}} = \{\alpha(x)q^{\mathcal{A}} \mid x \in V_C, q \in L^*\} \cup \{a^{\mathcal{A}} \mid a \in A\}$.

To see that $D^{\mathcal{A}}$ is finite, we first observe that the domain of each $\alpha(x)$ is a regular set ($\{p \mid xpa \in \mathcal{N}_C, a \in A\}$) and the range is finite. Now, for a regular set R call $\{p \mid qp \in R\}$ the suffix language of R with respect to string q. It is clear, that there are only finitely many suffix languages, since each corresponds to one state in the minimal finite automaton for R. But then also the number of partial functions "reachable" from an $\alpha(x)$ is finite, since the domain of $\alpha(x)q^{\mathcal{A}}$ is a suffix language of the domain of $\alpha(x)$.

We now show that given 1) the model $(\mathcal{A}, \alpha)$ satisfies all constraints in C.

- If $x \doteq a \in C$: $xa \in L(\mathcal{N}_C) \Rightarrow (\epsilon, a) \in \alpha(x)$. Now we know from 1) that no other pair is in $\alpha(x)$, i.e. $\alpha(x) = a^{\mathcal{A}}$.
- If $x \doteq y \in C$: Since x occurs only once in C, the only transition from x is (x, ϵ, y), thus $(x,p,a) \in \hat{\delta}_C$ iff $(y,p,a) \in \hat{\delta}_C$. We conclude that $(p,a) \in \alpha(x)$ iff $(p,a) \in \alpha(y)$.
- If $xf \doteq y \in C$: Let $(p,a) \in \alpha(x)f^{\mathcal{A}}$. Then $(x, fp, a) \in \hat{\delta}_C$. This implies that there is a state x' reachable with n ϵ-moves ($n \geq 0$) from x such that $x'f \doteq y' \in C$ and $(y', p, a) \in \hat{\delta}_C$, i.e. x' is the last state before f is consumed on a path consuming fpa. But now, since ϵ-moves on such a chain correspond to subsumption constraints (none of the variables in the chain is isolated) and since C is transitively closed for subsumption constraints, C has to contain a constraint $x' \sqsubseteq x$. But the last condition for normal form now tells us, that also $y' \sqsubseteq y$ is in C, implying $(y, \epsilon, y') \in \delta_C$. Hence, $(y,p,a) \in \hat{\delta}_C$ and $(p,a) \in \alpha(y)$.

  Conversely, let $(p,a) \in \alpha(y)$. Then $(y,p,a) \in \hat{\delta}_C$. From the construction also $(x, f, y) \in \delta_C$, hence $(fp, a) \in \alpha(x)$ and $(p,a) \in \alpha(x)f^{\mathcal{A}}$.
- If $x \sqsubseteq y \in C$: The simulation witnessing $\alpha(x) \sqsubseteq^{\mathcal{A}} \alpha(y)$ is simply the subset relation. Suppose $(p,a) \in \alpha(x)$. We conclude $(x,p,a) \in \hat{\delta}_C$, but also $(y, \epsilon, x) \in \delta_C$. Hence, $(y,p,a) \in \hat{\delta}_C$ and $(p,a) \in \alpha(y)$.

In order to show the other direction let us first show a property needed in the proof.

Lemma 1 *If $(\mathcal{A}, \alpha)$ is a model for presolved clause C and $(x,p,s) \in \hat{\delta}_C$, then $\alpha(s) \sqsubseteq^{\mathcal{A}} \alpha(x)p^{\mathcal{A}}$. (If $s = a$ let $\alpha(a) = a^{\mathcal{A}}$ for this purpose.)*

Proof. We show by induction over the definition of $\hat{\delta}_C$ that, given the condition above, there exists a simulation Δ in $\mathcal{A}$ such that $\alpha(s)\Delta\alpha(x)p^{\mathcal{A}}$.

1. $p = \epsilon$ and $x = y$: $\Delta = \mathbf{ID}$.
2. $p = \epsilon$ and $y \sqsubseteq x \in C$: since α is a solution, there exists a simulation Δ with $\alpha(y)\Delta\alpha(x)$ [$= \alpha(x)\epsilon^{\mathcal{A}}$].
3. $p = f$ and $xf \doteq y \in C$: $\Delta = \mathbf{ID}$, since $\alpha(x)f^{\mathcal{A}} = \alpha(y)$.
4. $p = \epsilon$ and $x \doteq s \in C$: $\Delta = \mathbf{ID}$, since $\alpha(x) = \alpha(s)$.
5. $p = qr$ and $(x,q,y) \in \hat{\delta}_C$ and $(y,r,s) \in \hat{\delta}_C$: by induction hypothesis there exist Δ_1 and Δ_2 such that $\alpha(y)\Delta_1\alpha(x)q^{\mathcal{A}}$ and $\alpha(s)\Delta_2\alpha(y)r^{\mathcal{A}}$.

   Let $\Delta = (\Delta_1 \cup \Delta_2)^*$ (the transitive closure of their union), then $\alpha(y)\Delta\alpha(x)q^{\mathcal{A}}$ and $\alpha(s)\Delta\alpha(y)r^{\mathcal{A}}$. But now, since $\alpha(y)r^{\mathcal{A}} \downarrow$ and Δ is a simulation, also $\alpha(y)r^{\mathcal{A}}\Delta\alpha(x)q^{\mathcal{A}}r^{\mathcal{A}}$. Hence, $\alpha(s)\Delta\alpha(x)(qr)^{\mathcal{A}}$. □

Now let us proof 3) $\Rightarrow$ 1) of the main theorem by contradiction.

3) $\Rightarrow$ 1):

Suppose 1) does not hold, but $(\mathcal{A}, \alpha) \models C$. Then there is a string $xpa \in L(\mathcal{N}_C)$ such that

Case 1: $xpb \in L(\mathcal{N}_C)$ where $a \neq b$.

From $(x,p,a) \in \hat{\delta}_C$ and $(x,p,b) \in \hat{\delta}_C$ we know with lemma 1 that $a^{\mathcal{A}} \sqsubseteq^{\mathcal{A}} \alpha(x)p^{\mathcal{A}}$ and $b^{\mathcal{A}} \sqsubseteq^{\mathcal{A}} \alpha(x)p^{\mathcal{A}}$. But this contradicts condition 1) for a simulation: $\alpha(x)p^{\mathcal{A}} = a^{\mathcal{A}} \neq b^{\mathcal{A}} = \alpha(x)p^{\mathcal{A}}$.

Case 2: $xpf \in L(\mathcal{N}_C)$.

As in case 1) we have $\alpha(x)p^{\mathcal{A}} = a^{\mathcal{A}}$. From $(x, pf, y) \in \hat{\delta}_C$ we get $\alpha(y) \sqsubseteq^{\mathcal{A}} \alpha(x)(pf)^{\mathcal{A}}$, which entails that $f^{\mathcal{A}}$ has to be defined for $\alpha(x)p^{\mathcal{A}}$, a contradiction.

This completes the proof. □