*J. Symbolic Computation* (1992) **13**, 441-461

# On Subsumption and Semiunification In Feature Algebras*

JOCHEN DÖRRE[†] AND WILLIAM C. ROUNDS[‡]

[†] *IBM Deutschland GmbH*
*Institute for Knowledge-Based Systems*
*P.O. Box 80 08 80*
*D-7000 Stuttgart 80, Germany*

[‡] *Artificial Intelligence Laboratory*
*EECS Department*
*University of Michigan*
*Ann Arbor, Michigan 48109, USA*

*(Received 23 April 1990)*

We consider a generalization of term subsumption, or matching, to a class of mathematical structures which we call *feature algebras.* We show how these generalize both first-order terms and the feature structures used in computational linguistics. The notion of term subsumption generalizes to a natural notion of algebra homomorphism. In the setting of feature algebras, unification corresponds naturally to solving constraints involving equalities between strings of unary function symbols, and semiunification also allows inequalities representing subsumption constraints. Our generalization allows us to show that the semiunification problem for finite feature algebras is undecidable. This implies that the corresponding problem for rational trees (cyclic terms) is also undecidable.

## 1. Introduction

Feature algebras are a generalization of a number of notions in artificial intelligence, especially knowledge representation schemata like frames and records; the feature structures in computational linguistics, which are used heavily in unification-based grammar systems, and the usual first-order terms in logic and logic programming, which are used as encodings of objects bearing information. They make explicit the notion of attribute and value in a particularly simple way. The simplicity and generality of the notion of feature algebra allows us to show easily that a case of the semiunification problem is undecidable,

*The inspiration for this paper came from work done at the Institut für maschinelle Sprachverarbeitung, University of Stuttgart, which the second author visited as a Gastprofessor in 1989. We would like to thank our colleagues there for providing such a stimulating atmosphere. The work was in part supported by ESPRIT Basic Research Action 3175 (DYANA).

settling one part of this formerly open problem. While this is the principal technical result, we would suggest that the algebraic problem-solving and mathematical modelling techniques associated with this class of structures comprise the major contribution of the paper.

To expand on this a bit, we think that the setting of the semiunification problem for feature algebras may be a more natural way to state the problem than for terms. We get both a natural presentation of both the cyclic and acyclic cases of the problem, and a short proof of undecidability in the cyclic case, using a reduction from the word problem for finite semigroups (Gurevich, 1966). Our technique does not apply to the original semiunification problem, as stated, for example, in (Leiß, 1988) or (Kfoury et al., 1989), because it makes essential use of the cyclic property. However, Kfoury, Tiuryn, and Urzyczyn (1990) have shown that the problem as stated for ordinary terms is also undecidable. Their proof uses a very different technique; namely, a reduction from the boundedness problem for Turing machines. Comparison of the two proofs suggests that the two cases of the problem are themselves very different; neither undecidability result implies the other. What seems to be an obstruction for an algorithm trying to decide an instance of the acyclic case is the so-called "occurs check", the need to detect the case when a solution would possibly become cyclic. So, the reduction in (Kfoury et al., 1990) does not need to use constants, and it fails for the cyclic case of the problem. Our reduction, on the other hand, must use constants, because otherwise there is always a trivial cyclic solution. Our proof thus shows that even if terms can be cyclic, the semiunification problem stays undecidable.

In natural-language applications, one needs circular structures, because one can have self-referential sentences like the Liar, as well as self-referential type descriptors (for example, the type of person who is self-employed.) Our results therefore have bearing if one models subsumption for feature terms ($\psi$-terms of Aït-Kaci (1986) ) as we do here. However, it may be possible to get by with a weaker notion in the natural language case. If one models subsumption not by using functions (see below), but by using relations, then the semiunification problem becomes decidable. This result appears in Dörre (1990).

Our paper is organized as follows. Section 2 contains basic definitions. Here we use notation borrowed from an unpublished draft by Gert Smolka. Our definition of subsumption owes much to his work on feature logic (Smolka, 1988), which is a version for arbitrary domains of the original Kasper-Rounds logic (Rounds & Kasper, 1986) for the domain of feature structures. Mark Johnson (1988) and Smolka had the idea to introduce these domains, and the present paper works out some of the consequences of this insight. The mathematical model provides us with some powerful new tools, in that the algebraic and model-theoretic techniques can be combined to prove what is actually a rather surprising undecidability result.

Before we get to the details of this result, we present in Section 3 some motivational material showing how feature algebras arise in the context of analyzing natural languages and programming languages. The reader who is only interested in the technical results can skip this section, which is presented informally. This motivation includes a way (using feature constraints) of modelling both natural language and ML-style type inferencing problems, sketching material from Shieber's dissertation (1989).

The undecidability results appear in Section 4, where we prove the undecidability of the semiunification problem for finite feature algebras, and in Section 5, where we reduce this decision problem to a more standard one: that for rational trees.

## 2. Feature algebras : Basic definitions

We begin by assuming two disjoint alphabets $L$ and $A$, called the sets of *features* and *atoms* respectively. Features are unary function symbols, and atoms are atomic constants in our interpretation. Generally we use the letters $f, g, h$ for features and $a, b, c$ for atoms.

A *feature algebra* $\mathcal{A}$ consists of a nonempty set $D^{\mathcal{A}}$ and a function $\cdot^{\mathcal{A}}$ defined on $L$ and $A$ such that

- If $f$ is a feature then $f^{\mathcal{A}}$ is a unary partial function[†] on $D^{\mathcal{A}}$;
- $a^{\mathcal{A}} \in D^{\mathcal{A}}$ for $a \in A$;
- If $a \neq b$ then $a^{\mathcal{A}} \neq b^{\mathcal{A}}$;
- No feature is defined on an atom.

NOTATION. We write function symbols on the right, so that $f(d)$ is written $df$. If $f$ is defined at $d$, we write $df \downarrow$, and otherwise $df \uparrow$. We use $p, q, s, t$ to denote strings of features, and if we write an equation $dp = eq$ it is intended that the appropriate composed function is defined.

EXAMPLES. The following are two canonical examples of feature algebras, though there are many others of interest. For other examples, see Sections 4.3 and 5.2.

- THE TERM ALGEBRA $\mathcal{T}(\Sigma, X)$. Let $\Sigma$ be a ranked alphabet of function symbols[‡], and $X$ a countable set of variables. The elements of $\mathcal{T}(\Sigma, X)$ are first-order terms over $\Sigma$ and $X$, together with the set $\Sigma$ itself. For the atoms of $\mathcal{T}(\Sigma, X)$ we take the elements of $\Sigma$; i.e., atoms are the function symbols of all arities. The features are the natural numbers $1, 2, \ldots$, and one extra feature FUN. The feature $i$ gives us the $i$th argument term of a term, if it exists, and the feature FUN gives us the topmost function symbol of a term. This feature is not defined on variables. For example, we have $(\sigma(x, \tau(a), b))2 = \tau(a)$, and $(\sigma(x, \tau(a), b))\mathrm{FUN} = \sigma$, where $\sigma$ has rank 3, and $\tau$ rank 1.
- THE FEATURE GRAPH ALGEBRA $\mathcal{F}$. The nonatomic elements of this algebra are pairs $(G, n)$, where $G$ is a (possibly infinite) directed graph, and $n$ is a node of $G$. Nodes are taken from a fixed countable set; say the integers. Each arc is labeled with an element of $L$, and no two outgoing arcs are labeled with the same element of $L$. Nodes with no outgoing arcs may optionally be labeled with elements of $A$. In this algebra, we interpret features $f$ as follows: let $(G, n)f$ be the graph $(G, nf)$, where $nf$ is the unique node of $G$ pointed to by the arc starting at $n$ and labeled by $f$, if there is such an arc, and if $nf$ is not labeled with an atom. If the node $nf$ is labeled with an atom $a$ then we define $(G, n)f = a$. Atoms in this algebra are thus just the elements of $A$ themselves.

This last example is a bit complex; we want it because it gives us an easy way of generalizing ordinary terms to the circular case, and, in the case where the graphs are finite, of picturing the feature structures of computational linguistics. One can think of

[†]These functions will also be called features.

[‡]Rank is the same as arity.

graph-node pairs $(G, n)$ as generalized terms, where $n$ identifies the root node or head of the term, or as transition graphs for automata, where $n$ is the initial state. (An example feature graph appears in Section 3.) In addition, $\mathcal{F}$ is an example of a general algebra which is canonical with respect to various solution properties; see Section 4.3 for the notion of "canonical".

In a general feature algebra, features are used to identify attributes of objects in the domain. Examples abound in computer science; in relational database systems, one has attributes *age, salary*, and so forth. Slots in frame formalisms are another class of examples. The atoms in a general feature algebra are like atomic constants, and retain the syntactic flavor which they have in the term algebra and the feature graph algebra. In a sense, atoms give us constant types. In Section 5 we will consider a generalization of feature algebras, where atoms are more liberally interpreted, but our undecidability results can be formulated for the present version.

Notice that any nonempty subset of a feature algebra containing interpretations for all the atoms is again a feature algebra. (We could allow atoms to be partially interpreted so that the above restriction was not necessary to make.)

We say that a feature algebra is finite if its domain is a finite set. Given a feature algebra $\mathcal{A}$ and a point $d \in D^{\mathcal{A}}$, the *extent* of $d$, written $Ext(d)$, is the set

$$\{dp^{\mathcal{A}} \mid p \in L^* \text{ and } dp^{\mathcal{A}} \downarrow\}.$$

We say that a feature algebra is *path-finite* if for each $d$, the set $Paths(d)$ is finite, where $Paths(d) = \{w \in L^* \mid dw^{\mathcal{A}} \downarrow\}$. The semiunification decision problem in Section 4 concerns finite feature algebras; the same problem for path-finite algebras is equivalent to the semiunification problem stated, say, in (Kfoury et al., 1989). The next definition introduces the key concept used to state these problems, and which is our real object of concern in this paper.

A *homomorphism* between two feature algebras $\mathcal{A}$ and $\mathcal{B}$ is a partial map $\gamma$ between the two domains satisfying

1. $(a^{\mathcal{A}})\gamma = a^{\mathcal{B}}$ for each atom $a$;
2. For any $d \in D^{\mathcal{A}}$ and $f \in L$, if $d\gamma \downarrow$ and $df^{\mathcal{A}} \downarrow$, then $df^{\mathcal{A}}\gamma = d\gamma f^{\mathcal{B}}$. (In particular, $d\gamma f^{\mathcal{B}}$ is defined.)

If $\gamma$ maps $\mathcal{A}$ to itself it is an *endomorphism*.

DEFINITION. Let $\mathcal{A}$ be a feature algebra. The *subsumption preorder* $\sqsubseteq$ on $\mathcal{A}$ is defined as follows:

$$d \sqsubseteq e \iff \text{there is an endomorphism } \gamma : d\gamma = e.$$

We say that $d$ *subsumes* $e$.

In our examples, one term subsumes a second iff the second is more instantiated than the first; stated another way, the first matches the second. A feature graph subsumes a second essentially when the second has more arcs, and when it makes more identifications than the first. (A feature graph identifies two paths when those two paths lead to a single node.)

## 3. Unification Grammars and Type Inference

Here we review the problems which naturally lead to the question of solving systems of constraints in feature algebras. The material on unification in natural language has a long history, and is reviewed in Shieber (1987). The connections between unification and type inference began with Hindley (1969) and Milner (1978). Henglein (1988) and Leiß (1988) showed a reduction of ML-style polymorphic type inference to semiunification for first-order terms. The paper by Kfoury et al. (1989) gives a proof that ML-style type inference with polymorphic recursion and the (acyclic) semiunification problem are recursively equivalent, as does Henglein (1989). Moshier (1988) showed for the first time how unification grammars from natural language studies could be applied to type inference in programming languages, and Shieber (1989) shows that in fact some weak forms of semiunification problems seem to occur in natural language itself, completing the connection.

Our presentation of this material is taken from Shieber's very readable 1989 thesis, which treats both the programming and natural language cases, using feature structures as the bearers of information. We begin with the ML-style polymorphic type inference problem. Consider the problem of defining polymorphically typed expressions, as in the program fragment *let* $f(x) = x$ *in* $f(f)(3)$. This is a standard example of polymorphism, where the identifier $f$, of polymorphic type $\forall\alpha.(\alpha \to \alpha)$, has been used twice, once as a function from $(INT \to INT)$ to $(INT \to INT)$, and once as a function from $INT$ to $INT$. Each of these uses is consistent with the polymorphic type, but incompatible with each other. The whole expression, however, is well-typed as an $INT$ with value 3.

Ignoring details of the *let* construct, and binding of variables, let us consider the usual type inference rule for functional application.

$$\frac{E \vdash f : A \to R \quad E \vdash a : A}{E \vdash f(a) : R}$$

It says the following: *Given a function $f$ of type $A \to R$ and an argument $a$ of type $A$ we can conclude the resulting expression $f(a)$ of being of type $R$ for a given environment* $E$ (a map assigning types to identifiers).

Instead of using this rule directly, however, we encode it using features as a grammar rule in the style of Shieber's PATR-II (1987).

$$\begin{gathered} Expr_R \longrightarrow Expr_F \text{ "(" } Expr_A \text{ ")" } : \\ F\ type\ constr \doteq \text{FUNCTION}; \\ F\ type\ arg \doteq A\ type\ ; \\ F\ type\ result \doteq R\ type\ ; \\ F\ env \doteq A\ env\ ; \\ R\ env \doteq A\ env\ . \end{gathered}$$

It consists of a context-free rule describing the form of expressions as well as an associated feature clause which must be satisfied at the appropriate node of the parse tree.[†] The occurences of the nonterminal *Expr* in the rule carry indices, which are best regarded

[†] In this framework nodes of a parse tree bear a richer structure than only a symbol label including in our case an encoding of the type of the dominated expression.

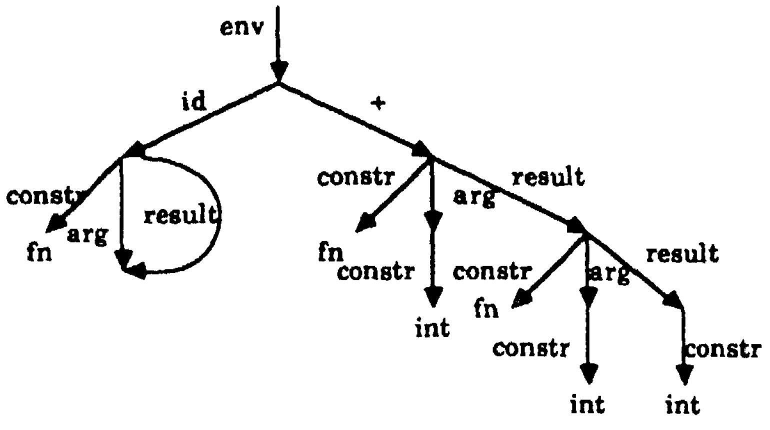

Figure 1. An environment.

as variables which are bound to the corresponding node of a parse tree if we apply the rule. We can thus require certain properties to hold at the different nodes using *feature constraints* over these variables. In the example, *type, constr, arg, result,* and *env* are features. The first three constraints enforce the matching conditions in the rule, namely that the type of $F$ is a function, indicated with the atom FUNCTION, whose argument and result types are identical to $A$'s type and $R$'s type, respectively. The last two constraints, using the feature *env*, encode the sharing of the environment $E$ among the three nodes.

To be more precise, we assume the nodes of a parse tree to be elements of a feature algebra $\mathcal{A}$ which interprets the features and atoms appearing in the constraints. For a given parse tree we get – induced by the grammar rule applications which constructed it – a collection of feature constraints over its nodes. Now, the feature algebra $\mathcal{A}$ has to satisfy this collection, i.e., for each "dotted equality" the left-hand side and the right-hand side have to be defined and denote identical elements. Without loss of generality we can think of the elements of $\mathcal{A}$ as *feature graphs*, like in our example in Section 2. The types of expressions as well as environments are encoded in these feature graphs.

For example, the graph in Figure 1 encodes an environment giving to the identifiers *id* and + the typings

$$id : (\forall\alpha)\alpha \rightarrow \alpha, \text{ and}$$
$$+ : INT \rightarrow (INT \rightarrow INT).$$

This environment, or symbol table, would be available at the point of application of the above rule, through recursive uses of the last two constraints, which pass environments up and down a tree. In this case we postulate constant definitions of symbols like + and *id*, and use them as feature names. Then to round out this simple grammar, we could introduce the rule

$$Expr_A \longrightarrow \omega \; (\omega \in \Sigma)$$

where $\Sigma$ is a prespecified set of identifier names like *id* and $+$, to be used both as terminal symbols and as features indexing into the environment.

To achieve ML-style type inferencing, this rule might be constrained by

$$A \; type \doteq A \; env \; \omega,$$

corresponding to the traditional type inference rule

$$\frac{E(\omega) = T}{E \vdash \omega : T}$$

And finally, we could have a rule

$$Expr_A \longrightarrow \iota \; (\iota \in N)$$

for integers, constrained by

$$A \; type \; constr \doteq \text{INT}.$$

We ignore the details of how to construct environments here; Shieber's thesis treats the problems of binding, and the typing of bound variables, in more detail, as does Moshier's. It will suffice here to suppose that environments are explicitly supplied with the start rule of the grammar by means of explicit constraints. Then this grammar, used schematically, allows the following typings:

- $id : \forall\alpha.\alpha \rightarrow \alpha$ ;
- $+ : INT \rightarrow (INT \rightarrow INT)$ ;
- $id(+) : INT \rightarrow (INT \rightarrow INT)$ ;
- $id(+(3)(id(4))) : INT$

Unfortunately, it does not allow $id(id)(3)$ to be well-typed, as in our problem above. This is because the two uses of the identity function are not consistent (fail to unify) with each other. An obvious solution to this problem is to relax the identity constraint

$$A \; env \; \omega \doteq A \; type$$

in the second rule, and replace it with a less restrictive one:

$$A \; env \; \omega \sqsubseteq A \; type \, .$$

This requires only that the type of a use be more instantiated than a type given by the original definition of the identifier. In the case of *id*, the original definition gives a type where the argument and result types are equal (Figure 1). It is easy to see that this matching corresponds to the existence of a feature graph homomorphism from the less instantiated type to the more instantiated one.

Continuing with Shieber's examples, we notice that the polymorphism problem above occurs in natural language processing as well. Consider coordinate constructions involving the conjunction "and", as in (1,2).

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

Example (2) is ungrammatical, so it would seem that the types of the coordinated phrases should be identical, and be the same as that of the whole coordinated complement of the verb "hire", which requires a noun-phrase object complement. But certain verbs, used very frequently, do not require this strict identity. The verb "become" allows either noun-phrase or adjective-phrase arguments, while "to be" allows prepositional and verb phrase arguments in addition. In fact, these arguments can be intermixed, as in (3,4).

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

(See (Shieber, 1989) for sources and further discussion of these examples.)

The "identity view" of coordinate conjunction might be expressed by a schematic rule like the following:

$$E ::= C \text{ AND } D$$
$$E \doteq C$$
$$E \doteq D$$

requiring that the type of the phrase $E$ be identical with that of both its constituents. However, the "polymorphic view" would suggest instead a rule like

$$E ::= C \text{ AND } D$$
$$E \sqsubseteq C$$
$$E \sqsubseteq D$$

which, given appropriate feature structures for the phrase types involved, and feature requirements of the verbs "to be" and "become" would allow the intermixed sentences to be grammatical.[†]

There is much more to the story of feature algebras. They are being used as a basis the formal semantics of the programming language LIFE, by Aït-Kaci and Podelski (1991). Here feature algebras appear generalized in an order-sorted framework. The reader can find another generalization and a characterization of the subsumption relation in our conference article (Dörre & Rounds, 1990). Here we again treat an order-sorted version, using the work of Smolka (1988), and a logical characterization of the subsumption relation extending the characterizations in (Rounds & Kasper, 1986). These results show explicitly how the subsumption relation corresponds to an information-theoretic ordering.

## 4. Undecidability results

We now turn to the technical results. We first prove our undecidablility results for feature algebras. In Section 5, we further reduce the problem to the semiunification problem for so-called rational terms. We have to leave unsolved, however, the existence of an algorithm to find path-finite solutions (this is essentially the original semiunification problem; it has just been shown undecidable by Kfoury et al. (1990) ).

[†]Shieber uses a technically different notion of subsumption than we do.

## 4.1. CONSTRAINTS

To begin, we assume that we have an infinite set $X$ of *variables* written as $x, y$ and so forth. Then a *constraint* is a piece of syntax having one of the forms

$$xp \doteq yq, \; xp \doteq a, \; xp \sqsubseteq yq$$

where $p$ and $q$ are paths, i.e., elements of $L^*$, and $a \in A$. A *clause* is a finite set of constraints of the above forms. Given a feature algebra $\mathcal{A}$, an *assignment* is a mapping $\alpha$ of variables to the elements of $\mathcal{A}$. The *solutions* in a feature algebra $\mathcal{A}$ of a constraint $\phi$ are those assignments $\alpha$ which satisfy the constraint in the expected way:

$$\begin{array}{lll} (\mathcal{A}, \alpha) \models xp \doteq yq & \text{iff} & \alpha(x)p^{\mathcal{A}} = \alpha(y)q^{\mathcal{A}} \; ; \\ (\mathcal{A}, \alpha) \models xp \doteq a & \text{iff} & \alpha(x)p^{\mathcal{A}} = a^{\mathcal{A}} \; ; \\ (\mathcal{A}, \alpha) \models xp \sqsubseteq yq & \text{iff} & \alpha(x)p^{\mathcal{A}} \sqsubseteq \alpha(y)q^{\mathcal{A}}. \end{array}$$

An assignment is a solution in $\mathcal{A}$ of a clause $C$ iff it is a solution for all constraints in $C$. A clause is *satisfiable* if it has a solution in some feature algebra, and *finitely satisfiable* iff it has a solution in some finite feature algebra.

We now state the *semiunification problem* for feature clauses:

Given finite $L$ and $A$, and a clause $C$ over these alphabets, is $C$ satisfiable?

The *finite* semiunification problem asks the same question about finite satisfiability.

THEOREM 1. *Both the semiunification problem and the finite semiunification problem are undecidable.*

*Proof.* We use a result due to Gurevich (1966), which is that the word problem for finitely generated finite semigroups is undecidable. We reduce this problem to the finite semiunification problem. (The same proof works to reduce the well-known word problem for arbitrary semigroups to the arbitrary semiunification problem.) The original use of this technique is due to Schmidt-Schauß (1989), who used it to show that subsumption in KL-ONE is undecidable (note, however, that this use of subsumption is not the same as ours.)

The word problem for finite semigroups is the following. Given a finite alphabet $\Sigma$, consider the class $\Gamma$ of finite semigroups finitely generated by $\Sigma$. Let $E$ be a finite set of equations $s_i = t_i$, $i = 1, \ldots, n$, where the $s_i$ and $t_i$ are nonempty strings over $\Sigma$. Let $s = t$ be another such equation; then the word problem is to determine whether or not every semigroup in $\Gamma$ satisfying all the equations in $E$ also satisfies the equation $s = t$.

Suppose therefore that a system of equations $E$ over some finite alphabet $\Sigma$ is given, together with a test equation $s = t$. We first choose our feature alphabet to be the alphabet

$$L = \Sigma \cup \{k\}$$

where $k$ is not in $\Sigma$. We also choose two distinct constants $a$ and $b$.

We construct a clause $C_E$ from $E$ as follows:

1. For each $f \in \Sigma$, add the constraint $x \sqsubseteq xf$ to $C_E$;
2. For each equation $s_i = t_i$ in $E$, add a constraint $xs_i \doteq xt_i$;

3. Add the constraints $x \sqsubseteq y$, $ysk \doteq a$, and $ytk \doteq b$ to $C_E$.

We claim that $C_E$ has a solution in some finite feature algebra if and only if there is a finite semigroup satisfying every equation in $E$ but not the equation $s = t$.

To verify this claim, we first establish two easy lemmas.

LEMMA 1. *Let $\mathcal{A}$ be a feature algebra satisfying the constraints (1) for each element $f \in \Sigma$, and let $\alpha$ be a solution. Then $\alpha(x)p^{\mathcal{A}} \downarrow$ for any $p \in \Sigma^+$.*

*Proof.* First let us unclutter our notation by dropping the explicit mention of the assignment $\alpha$ and the superscripts $\mathcal{A}$. To prove the lemma, use induction on the length of $p$. If this length is 1, the constraints themselves give the result. In the inductive case, $p = fq$ for some string $q$. We know that there is an endomorphism $\gamma$ such that $x\gamma = xf$. By the definition of homomorphism, then, $xq\gamma = x\gamma q = xfq = xp$. Therefore in particular $xp \downarrow$ . This completes the proof.

In the next lemma, we continue with the uncluttered notation.

LEMMA 2. *Under the same hypotheses on the feature algebra, let $u, v$ be any strings in $\Sigma^*$ and let $p \in \Sigma^*$. If $xu = xv$ in $\mathcal{A}$, then also $xpu = xpv$ in $\mathcal{A}$.*

*Proof.* Again by induction on the length of $p$. The case of length 0 is trivial. Suppose then that $p = fq$ and that $xqu = xqv$. Since $x \sqsubseteq xf$, we have that as in Lemma 1 $xq\gamma = xp$ for some homomorphism $\gamma$. We have $xqu\gamma = xq\gamma u = xpu$, and $xqv\gamma = xq\gamma v = xpv$. But $xqu = xqv$, so this completes the proof of the lemma.

Now let us return to the main proof. Suppose first there is a finite algebra $\mathcal{A}$ satisfying (1), (2), and (3) of $C_E$. Construct a finite semigroup $S$ as follows. Let $J$ be the set $\{xp^{\mathcal{A}} \mid p \in \Sigma^+\}$. Then $J$ is a subset of $D^{\mathcal{A}}$ and so is finite. The collection of all $p^{\mathcal{A}}$ restricted to $J$ is a finite semigroup $S$ under composition, where $p$ is an arbitrary nonempty composition of interpreted features in $\mathcal{A}$. Now the constraints (2), together with Lemma 2, imply that $S$ satisfies the equations in $E$. (The conclusion of Lemma 2 just says that $S$ works in exactly the right way as a semigroup of functions.) Now if $S$ satisfied the equation $s = t$, we would have in particular that $xs = xt$ in $\mathcal{A}$. But now $x \sqsubseteq y$ in $\mathcal{A}$, which implies that $ys = yt$ in $\mathcal{A}$, impossible in view of the constraints (3). Thus the constructed semigroup satisfies the equations in $E$ but not the equation $s = t$. This completes the first part of our claim.

For the second part, suppose there is a finite semigroup $S$ satisfying everything in $E$ but not $s = t$. Construct a finite feature algebra $\mathcal{A}$ as follows. First adjoin an identity $e$ to $S$ if one is not there already. We will still call the result $S$. Now let $S^1$ and $S^2$ be isomorphic disjoint copies of $S$. Let $a$ and $b$ be two extra atomic elements. Then $D^{\mathcal{A}} = S^1 \cup S^2 \cup \{a, b\}$. Let us write $f^1$ and $f^2$ for the elements corresponding to $f$ in $S^1$ and $S^2$, respectively, and let a string of symbols with a superscript, $s^i$, denote the product of the corresponding elements in $S^i, i = 1, 2$. Now, for each $f \in \Sigma$ define $(s^i)f^{\mathcal{A}}$ to be the product $s^i \cdot f^i$ in $S^i, i = 1, 2$. Let $s$ and $t$ be the strings in the given test equation $s = t$, and let $s^2$ and $t^2$ be their values in $S^2$. Then set $s^2k^{\mathcal{A}} = a$ and $t^2k^{\mathcal{A}} = b$. Since $s^2 \neq t^2$, this is possible. Now we check that $\mathcal{A}$ satisfies the constraints of $C_E$. Map $x$ to $e^1$, and $y$ to $e^2$. For $u$ in $S^1$, and $f \in \Sigma$, the mapping sending $u$ to $f \cdot u$ in $S^1$ is an $\mathcal{A}$-endomorphism mapping $e$ to $f$, so (1) is satisfied. (2) is satisfied because $S^1$ satisfies $E$. As for (3), the mapping sending $u^1 \in S^1$ to its copy $u^2 \in S^2$ is an $\mathcal{A}$-endomorphism verifying $x \sqsubseteq y$, and we have already satisfied the last two constraints. This proves the theorem.

### 4.2. Related undecidable problems for feature algebras

Here we state two further undecidability results following from the same technique that we used in Theorem 1. Notice first that if a feature constraint is satisfiable, say in a finite algebra, then it is satisfiable in an infinite one obtained by adding arbitrary elements to the domain, unconnected to the original elements. We might, however, be concerned with the existence of an infinite "reachable" solution. So we define the ***cardinality of a solution*** $\alpha$ of a clause $C$ to be the cardinality of the set

$$\bigcup\{Ext(\alpha(x)) \mid x \text{ is a variable of } C\}.$$

We will show that it is undecidable whether or not a clause $C$ has a solution of infinite cardinality. To do this we use a classical word problem undecidability result due to Adjan (1955) and also to Rabin (1958). To state the result fully we need some definitions. Say that a class of groups $\Gamma$ is *invariant* iff for any groups $G$ and $H$, if $G \in \Gamma$ and $H$ is isomorphic to $G$, then $H \in \Gamma$. The class $\Gamma$ is *nontrivial* iff there is a group in $\Gamma$ and a group not in $\Gamma$. Finally, $\Gamma$ is *hereditary* iff whenever $G$ is in $\Gamma$, and $H$ is a subgroup of $G$, then $H$ is in $\Gamma$. The Adjan-Rabin theorem, a kind of Rice's theorem for word problems on groups, reads as follows.

THEOREM 2. (ADJAN-RABIN) *Let $\Gamma$ be an invariant, nontrivial, and hereditary class of groups. Then there is no algorithm to determine, given a finite presentation of a group $G$, whether or not $G \in \Gamma$.*

For this theorem, a presentation is just a finite set of words in the generator symbols and their inverses, which intuitively rewrite to the identity. For groups, we can consider the smallest normal subgroup of the free group over the generators, containing the words in the presentation. The quotient group is the group $G$ presented by the given set of words.

Now for our application, we have to consider a modified version of the Adjan-Rabin theorem, stated for monoids. This time, a presentation is a set of equations in the generators, as in the word problem for semigroups above. However, we also allow an equation $s = e$, where $s$ is a string of generator symbols, and $e$ denotes the identity of the monoid. The monoid $M_E$ presented by the set of equations $E$ is the collection of equivalence classes of words under the congruence relation $\equiv$, which is the reflexive, transitive closure of the usual rewriting relation on $\Sigma^*$ given by the equations $E$. Our modified restatement of the Adjan-Rabin theorem is as follows.

LEMMA 3. *Let $\Gamma$ be an invariant class of monoids, such that the class of groups in $\Gamma$ is a nontrivial and hereditary class of groups. Then there is no algorithm to determine, given a finite presentation of a monoid $M$, whether or not $M \in \Gamma$.*

*Proof.* We simply reduce the Adjan-Rabin decision problem to this one. Given a group presentation $P$, construct the set of equations $E_P$ (over the expanded alphabet containing generator symbols and their inverses.) Add the equations which state that $ff^{-1} = e$ for the generators $f$. These equations induce a congruence relation (on the expanded alphabet) in the usual way, and the congruence classes have a group structure; this group (say $G$) is the monoid defined by the presentation, and is of course isomorphic to the group defined by the group presentation. Let $\Gamma'$ be the set of groups in $\Gamma$. Then $G$ is in $\Gamma$ if and only if it is in $\Gamma'$, so the Adjan-Rabin theorem gives us our result.

Now let us state our application.

THEOREM 3. *It is undecidable whether or not a clause $C$ has a solution of infinite cardinality.*

*Proof.* We take the class $\Gamma$ to be the class of finite monoids in the modified Adjan-Rabin lemma. Clearly $\Gamma$ satisfies the hypotheses of the lemma. It follows that, given a set of equations in the generators of a monoid, it is undecidable if the presented monoid is finite or not. Given the presentation $E$, we build a constraint system $C_E$ as in Theorem 1, but more simply. $C_E$ consists of the constraints (2) used to define the monoid together with the constraints (1).

Let $M_E$ be the monoid defined by the presentation $E$. We claim that $M_E$ is infinite if and only if there is a solution of $C_E$ of infinite cardinality. To see this, proceed as in Theorem 1. Suppose there is a solution to $C_E$ of infinite cardinality; that is, there is a feature algebra $\mathcal{A}$ and an assignment $\alpha$ such that $J = Ext(\alpha(x))$ is infinite, where $x$ is the single variable of $C_E$. Now $J$ is by definition

$$\{\alpha(x)p^{\mathcal{A}} \mid p \in L^* \text{ and } \alpha(x)p^{\mathcal{A}} \downarrow\}.$$

Since this set is infinite, the collection of all $p^{\mathcal{A}}$ restricted to $J$ must be an infinite monoid $M$ under composition, where $p$ is an arbitrary nonempty composition of interpreted features in $\mathcal{A}$. As in Theorem 1, $M$ satisfies the equations $E$. But then $M$ is a homomorphic image of $M_E$, because $M_E$ is initial in the variety of monoids generated by $\Sigma$ and satisfying the equations $E$. So $M_E$ is infinite.

Conversely, if $M_E$ is infinite, we make it into a feature algebra as in the second half of the proof of Theorem 1. If we let $\alpha(x)$ be the identity element of $M_E$, then it is straightforward to check that this provides a solution to $C_E$ of infinite cardinality. This completes the proof of Theorem 3.

We close the section with a strengthening of Theorem 1.

THEOREM 4. *It is undecidable whether or not a given clause has a solution in a finite feature algebra, even when subsumption homomorphisms are required to be injective.*

*Proof.* We use the undecidability of the word problem for finite groups (Slobodskoi, 1981), and proceed as in the proof of Theorem 2, using the same reduction. (Note that we are assuming the equational presentation of a group word problem, so that $ff^{-1} = e$ is always an equation.) We need to check that the proof works when all morphisms are required to be injective.

As above, suppose there is a finite feature algebra $\mathcal{A}$ satisfying (1), (2), and (3) of $C_E$. Construct a finite monoid $S$ as follows. Let $J$ be the set $\{xp^{\mathcal{A}} \mid p \in \Sigma^*\}$. Then $J$ is a subset of $D^{\mathcal{A}}$ and so is finite. The collection of all $p^{\mathcal{A}}$ restricted to $J$ is a finite monoid $S$ under composition, where $p$ is an arbitrary (possibly null) composition of interpreted features in $\mathcal{A}$. We need to show that in fact $S$ is a finite group.

To do this, we will show that for each function $f^{\mathcal{A}}$ in $S$, where $f$ is a feature, there is a natural number $m$ such that $f^m$ is the identity function $e$ on $J$. This element is the required inverse of $f$, and since $S$ is generated by the $f$ functions, this will suffice.

We show that for each function $f$, there is an integer $m$ such that $xf^m = x$, where $x$ is that element of $D^{\mathcal{A}}$ interpreting the variable $x$ of $C_E$. By lemma 2, this equation will hold for any element of J reachable from $x$, which is what we want. Now since $J$ is finite, there are integers $j < k$ such that $xf^j = xf^k$. Since $x \sqsubseteq xf$, we have for some injective morphism $\gamma$, $x\gamma = xf$. By induction,

$$x\gamma^j = xf^j = xf^k = x\gamma^k$$

whence $x\gamma^{k-j}\gamma^j = x\gamma^j$. But $\gamma^j$ must be injective, so $x\gamma^{k-j} = x$. Thus

$$xf^{k-j} = x\gamma^{k-j} = x$$

so we may choose $m = k - j$. This completes the proof that $S$ is a finite group.

Now we may proceed as in the proof of Theorem 1. The first part of the reduction goes just as in that theorem. For the converse, notice if $S$ is a finite *group* satisfying the equations $E$, then the morphisms defined in the second part of the proof of Theorem 1 are all injective. This completes the proof of Theorem 4.

A consequence of this theorem, as indicated in the next sections, is that the semiunification problem for rational terms remains undecidable even if all "matching substitutions" are required to be one-to-one. We do not know if this result holds for first-order terms or not.

## 4.3. Canonical algebras

We say that a feature algebra $\mathcal{B}$ is *canonical* for arbitrary solutions iff for every clause $C$, we have that $C$ is satisfiable if and only if there is a solution to $C$ in $\mathcal{B}$. Similarly, we define an algebra $\mathcal{B}$ to be canonical for finite solutions iff for every $C$, we have that $C$ is finitely satisfiable if and only if $C$ is satisfiable in $\mathcal{B}$. Note that in this case we do not require $\mathcal{B}$ to be finite. However, the elements of $\mathcal{B}$ will usually have finite representations.

In this section we will show that the algebra of finite feature graphs is canonical for finite solutions. In the next section, we will show that the algebra of rational trees also has this property. An obvious corollary of Theorem 1 is thus that the semiunification problem for these special feature algebras is undecidable.

Recalling the definition of Section 2, we define the finite feature graph algebra $\mathcal{F}_0$. The nonatomic elements consist of pairs $(G, n)$, where $G$ is a *finite* directed graph, and $n$ is a node of $G$. Again, nodes are taken from a fixed set $N$. Each arc is labeled with an element of $L$, and no two outgoing arcs are labeled with the same element of $L$. Nodes with no outgoing arcs may optionally be labeled with elements of $A$. As in the full feature graph algebra, we interpret features $f$ as follows: let $(G, n)f$ be the graph $(G, nf)$, where $nf$ is the unique node of $G$ pointed to by the arc starting at $n$ and labeled by $f$, if it exists, and if it is not labeled with an atom. If the node $nf$ is labeled with an atom $a$ then we define $(G, n)f = a$. Atoms in this algebra are the elements of $A$ themselves. (From now on, we will assume that $A$, the set of atoms, is finite.)

One can think profitably of the elements $(G, n)$ of $\mathcal{F}_0$ as (disconnected) transition graphs for finite state machines, where $n$ is the initial state. (One can also get a canonicity result for an algebra of graphs where every node is reachable from the initial one.)

THEOREM 5. *The feature algebra $\mathcal{F}_0$ is canonical for finite solutions.*

*Proof.* Let $C$ be a clause. Put $C$ into *standard form* by adding new variables and constraints as necessary. In the standard form, each constraint has one of the forms

$$x \doteq yf,\ x \doteq a,\ x \sqsubseteq y,\ x \doteq y$$

where $x$ and $y$ are variables, $a$ is a constant, and $f$ is a feature symbol. It will suffice to show that $\mathcal{F}_0$ is canonical for standard-form clauses. Suppose therefore that there is a solution $\alpha$ to $C$ in some algebra $\mathcal{A}$. Define a graph $G^{\mathcal{A}} \in \mathcal{F}_0$ as follows. For the nodes, take the set $D^{\mathcal{A}}$ (in reality, an isomorphic copy of $D^{\mathcal{A}}$ in the fixed node set $N$.) Put an arc $f$ from the node $d$ to node $d'$ just in case $df^{\mathcal{A}} = d'$. Define the solution $\beta(x)$ to be $(G^{\mathcal{A}}, \alpha(x))$ if $\alpha(x) \neq a^{\mathcal{A}}$ for an atom $a$; otherwise let $\beta(x) = a$. It is straightforward to check that $\beta$ satisfies all equational constraints of $C$. To show that $\beta$ satisfies all subsumption constraints of the form $x \sqsubseteq y$, proceed as follows. Let $\gamma$ be an endomorphism of $\mathcal{A}$. Then $\gamma$ induces an endomorphism $\gamma_0$ of $\mathcal{F}_0$ using the equations

$$(G^{\mathcal{A}}, d)\gamma_0 = \begin{cases} a & \text{if } d\gamma = a^{\mathcal{A}} \text{ for some } a \in A; \\ (G^{\mathcal{A}}, d\gamma) & \text{otherwise.} \end{cases}$$

Since $\alpha$ verifies $x \sqsubseteq y$, we have $\alpha(x) \sqsubseteq \alpha(y)$ in $\mathcal{A}$. Let $\gamma$ witness this relation in $\mathcal{A}$; clearly then $\beta(x)\gamma_0 = \beta(y)$, which is what we needed.

Conversely, suppose there is a solution $\beta$ to $C$ in $\mathcal{F}_0$. Define a finite feature algebra $\mathcal{A}$ as follows: take

$$D^{\mathcal{A}} = A \cup \{(G, m) \mid (\exists n)((G, n) = \beta(x) \text{ for some variable } x \text{ of } C)\}.$$

$\mathcal{A}$ is to be a finite subalgebra of $\mathcal{F}_0$; so we define $(G, m)f^{\mathcal{A}} = (G, mf)$ if $mf$ is nonatomic, and $(G, m)f^{\mathcal{A}} = a$ if $mf$ is labeled with the atom $a$. We can then take the solution $\alpha = \beta$. This completes the proof of the theorem.

## 5. The semiunification problem for rational trees

Theorem 1 also implies that the semiunification problem for rational trees is undecidable. Before we state this problem, we say what these trees are, and discuss notations for them. Then we state the semiunification problem (using the presentation in (Leiß, 1988)), and prove that it is undecidable using a series of reductions from the finite semiunification problem for feature algebras. Most of this is straightforward, but it shows precisely how to relate feature algebras with terms, encoding feature information using argument positions.

### 5.1. Rational trees: definition and notation

Let $\Sigma$ be an ordinary finite ranked alphabet, and $X$ a countable set of variables. We consider $\Sigma$-labeled, ordered infinite trees. Each node labeled with an $n$-ary operator symbol has $n$ descendants. Leaf nodes may be labeled with 0-ary symbols, or with variables. We say that a tree is *rational* if it has only finitely many nonisomorphic subtrees. Rational trees are also called cyclic terms. Let $RT(\Sigma, X)$ denote the set of rational trees.

There are a number of differing notations for representing these trees; we use one like that used by Colmerauer (1988). We explain the notation informally.

Consider the equation

$$x = \sigma(x) \tag{1}$$

where $\sigma$ is a unary function symbol. This equation defines a rational unary tree consisting of one infinite path with each node labeled by $\sigma$. We also can use the notation $\mu t.\sigma(t)$, a kind of fixed-point formula, to be the unique tree $t$ such that $t = \sigma(t)$, up to isomorphism, of course. This notation takes into account the fact that in equation (1) the variable $x$ really occurs bound. Now we also notice that the equational representation of a tree need not be unique, as in (2, 3).

$$x = \sigma(y) \tag{2}$$
$$y = \sigma(x). \tag{3}$$

Here, the system defines the same term for $x$ as does (1). Without getting into details, it is fairly clear that we can always decide if two equation systems represent the same rational tree.

A more complex example is (now $\sigma$ is a binary function symbol):

$$x = \sigma(z, x_1) \tag{4}$$
$$z = \sigma(z, x_1). \tag{5}$$

In this case, there is a free variable $x_1$ involved; but the variables $x$ and $z$ are bound. The equations (4, 5) are equivalent to the single equation

$$x = \sigma(x, x_1). \tag{6}$$

The tree represented by these equations has leaf nodes labeled with the variable $x_1$. It may also be denoted $\mu t.\sigma(t, x_1)$ which makes explicit the bound variable.

We will generally use *solved-form* systems of equations as in (4,5) to define rational trees. In this form, only variables occur on the left side of a system, and no variable will occur there more than once. Such variables will be considered bound in the system.

## 5.2. Rational trees as a feature algebra

We make $RT(\Sigma, X)$ into a feature algebra just as in the first example of Section 2. Once again, feature $i$ picks out the $i$-th immediate subtree of a given tree. If $\Sigma$ is finite then so is the feature alphabet. We will be interested especially in the case where $\Sigma$ has just one $n$-ary function symbol, for some $n \geq 1$, and a finite collection $A$ of 0-ary atomic symbols. We will call the corresponding feature algebra $\mathcal{R}_n$. In this case the feature FUN is uninteresting (!), so we do not define it.

The notion of *substitution* makes sense for rational trees as well as for ordinary terms. Substitutions are specified by assigning rational trees to variables in the usual way; these assignments then extend to the full domain $RT(\Sigma, X)$ as expected. We will use the letters $R, S, \ldots$ for substitutions. If $t$ and $u$ are in $RT(\Sigma, X)$, we say that $t$ *matches* $u$, written $t \leq u$, if there is a substitution $T$ such that $T(t) = u$.

EXAMPLE 1. Consider the solved-form equations

$$x = \sigma(z, x_1) \tag{7}$$
$$z = \sigma(x, x_2). \tag{8}$$

Here, the term defined as the value of $x$ is different from the term defined as the value for $z$, because $x_1$ and $x_2$ are differing free variables. But if we make the substitution $T(x_1) = x_2$ and $T(x_2) = x_1$ in (7,8), we get a system

$$\begin{aligned} x &= \sigma(z, x_2) \qquad (9)\\ z &= \sigma(x, x_1). \qquad (10)\end{aligned}$$

in which the equation for $z$ (10) defines the same term as equation (7) in the previous pair (7,8). So if $t$ is defined to be the solution for $x$ in (7,8), and $u$ is the solution for $z$ in the same system, then $t$ matches $u$.

REMARK 1. In $\mathcal{R}_n$, as well as in $RT(\Sigma, X)$, notice that $t \leq u$ if and only if $t \sqsubseteq u$. Further, this remark holds when feature morphisms and substitutions are injective.

## 5.3. THE SEMIUNIFICATION PROBLEM FOR RATIONAL TREES

Moving to the semiunification problem itself once again, we first recall the statement of the semiunification problem for ordinary terms (Henglein, 1988).

Consider a system $C$ of equations and inequalities between first-order terms, using different inequality symbols $\leq_1, \ldots, \leq_m$. We say that a substitution $R$ is a *semiunifier* for $C$ if

- $R(s) = R(t)$ for each equation $s = t$ in $C$;
- There are *residual substitutions* $T_1, \ldots, T_m$ such that $T_i(R(s)) = R(t)$ whenever $s \leq_i t$ is in $C$.

For rational trees, we will have to make a slightly different presentation, because each tree is itself given by solved-form equations as above. The details of this presentation are straightforward; the system $C$ will be a set of equations and inequalities between variables, each of which is the leading variable of the appropriate solved-form system of equations.[†]

The semiunification problem for rational trees is to decide, for a finite system $C$, whether or not it has a semiunifier.

EXAMPLE 2. Consider the system consisting of one inequality $x \leq \sigma(x)$. (Here $\sigma$ is a unary function symbol.) We can take $R$ to be the identity substitution and $T$ to be the substitution $x \mapsto \sigma(x)$.

EXAMPLE 3. The reverse system $\sigma(x) \leq x$ has no semiunifier in acyclic terms, but does have one in rational trees, namely $R$, mapping $x$ to $\mu t.\sigma(t)$. In this case the residual substitution is the identity.

## 5.4. THE REDUCTIONS

From now on, we will assume that our feature alphabet $L$ and the atom alphabet $A$ are finite, and that $L$ has $n$ symbols. We also fix the set $X$ of variables. $X$ will be used both to state feature constraints and to serve as the variable set in rational trees.

[†]It will actually turn out that the semiunification problem is undecidable, even if cyclic terms do not appear in the presentation of the constraint system $C$, so we do not need to worry about the official presentation.

For our first reduction, we take a system $C$ of feature constraints and, as in Theorem 5, put it into *standard form* where each constraint looks like one of

$$x \doteq yf,\ x \doteq a,\ x \sqsubseteq y,\ x \doteq y$$

where $x$ and $y$ are variables, $a$ is a constant, and $f$ is a feature symbol. It is clear that this transformation preserves the finite solution property, and from now on, clauses will be in this form.

Next, let $n$ be the size of $L$. We say that a feature algebra is $n$-ary iff for each $d$ in the domain, either $d$ has all $n$ features defined, or $d$ has no features defined. Notice that $\mathcal{R}_n$ is $n$-ary.

LEMMA 4. *A system of constraints over $A$ and $L$ has a solution in a finite feature algebra if and only if it has a solution in an $n$-ary finite feature algebra.*

*Proof.* The "if" direction is trivial, so suppose that $C$ has a solution in a finite feature algebra. For each element $d$ with a feature defined at it, suppose that the feature $g$ is not defined. Adjoin a distinct new element $m(d,g)$ to $D$, and let $dg = m(d,g)$ in the new algebra. The new algebra is finite and $n$-ary. Clearly the equational constraints in $C$ still hold in the new algebra. It is also straightforward to check that each morphism mapping $d$ to $e$ can be extended to a morphism of the new algebra, so that subsumption constraints still hold. This completes the proof.

LEMMA 5. *Let $C$ be a standard-form clause over $A$ and $L$. Then we may effectively find a standard-form clause $C_n$ over the feature alphabet $I_n = \{1,\ldots,n\}$ such that $C$ has a solution in a finite $n$-ary algebra over $A$ and $L$ iff $C_n$ has a solution in $\mathcal{R}_n$, where the constant symbols are taken from $A$.*

*Proof.* Let $C$ be a standard-form clause over the finite alphabets $A$ and $L$. Let $L = \{f_1,\ldots,f_n\}$. Replace each occurrence of a feature $f_i$ occurring in constraints of $C$ by the feature $i$. This defines the clause $C_n$.

Suppose that $C$ has a solution $\alpha$ in a finite $n$-ary algebra $\mathcal{A}$. Build a finite feature algebra $\mathcal{A}_1$ isomorphic to $\mathcal{A}$ as follows. Replace the 0-ary nonconstant elements in $D^{\mathcal{A}}$ by distinct variables taken from $X$ and (for safety) different from the variables used in the constraint $C$. (We also may assume that the domain $D^{\mathcal{A}}$ is disjoint from $X$.) By abuse of notation, let $\alpha$ be the solution to the clause $C$ in $\mathcal{A}_1$. We construct a solution $\beta$ to the constraint $C_n$ in $\mathcal{R}_n$. To define $\beta(x)$, consider $\alpha(x)$. The set $Ext(\alpha(x))$ is the set of states of a finite-state machine with initial state $\alpha(x)$, transitions from the alphabet $L$, and with terminating states (no outgoing transitions) labeled with variables or constants. Unroll this machine into an infinite tree in the standard manner, labeling the interior nodes with the $n$-ary function symbol $\sigma$, and replacing the arc label $f_i$ with the arc label $i$. In this manner we get a rational tree in $\mathcal{R}_n$, and this tree will be $\beta(x)$. More generally, we can take any element $d$ in the domain of $\mathcal{A}_1$ and unroll it into a rational tree $U(d)$; thus in particular $\beta(x) = U(\alpha(x))$. It is not hard to check that $\beta$ satisfies the equality constraints of $C_n$, so let us check the subsumption constraints. We show that if $d \sqsubseteq e$ in $\mathcal{A}_1$, then $U(d) \sqsubseteq U(e)$ in $\mathcal{R}_n$. Since the subsumption constraints are of the form $x \sqsubseteq y$, this will suffice. By Remark 1, we need only show that if $d \sqsubseteq e$, then $U(d) \leq U(e)$. Suppose $\gamma$ is a morphism taking $d$ to $e$. Let $z$ be a 0-ary nonatomic element of $Ext(d)$; then $\gamma$ is defined at $z$. Consider the substitution $T(z) = U(z\gamma)$. We have $T(U(d)) = U(e)$, as desired.

Conversely, suppose $\beta$ is a solution to $C_n$. Define a finite $n$-ary algebra $\mathcal{A}$ as follows; For $D^{\mathcal{A}}$ take $A$ together with all possible subtrees of $\beta(x)$, as $x$ ranges over the variables of the constraint $C_n$. Since the trees are rational, $D$ is finite. Let $uf_i^{\mathcal{A}} = ui$ if this is defined in the tree algebra. Then obviously $\alpha(x) = \beta(x)$ is a solution to $C$, and $\mathcal{A}$ is $n$-ary because $\mathcal{R}_n$ is. This completes the proof.

A consequence of the preceding lemma is that, modulo a slight change of feature alphabet, the algebra of rational trees is canonical for finite solutions. Also, it is easy to check that the lemma holds when the morphisms and substitutions are injective. But now let us continue with our final reduction.

LEMMA 6. *Let $C_n$ be a standard-form system over $A$ and $I_n$. Then we may effectively transform $C_n$ into a presentation $P_n$ of the semiunification problem for rational terms, such that $C_n$ has a solution in $\mathcal{R}_n$ iff $P_n$ has a semiunifier.*

*Proof.* Let $C_n$ be a standard-form feature clause. We construct the presentation $P_n$ as follows: For each constraint of the form $x \doteq a$ or $x \doteq y$ of $C_n$, add the same equation to $P_n$. For each subsumption constraint $x \sqsubseteq y$, add an inequality $x \leq_q y$ to $P_n$, where $q$ is a new integer each time. Let $\phi$ be a constraint in $C_n$ of the form $x \doteq yi$. Then for $j \neq i$ let $x(\phi, j)$ be a new variable, distinct from all variables for other constraints and from the variables in the clause $C$. Add the equation

$$y = \sigma(x(\phi,1), \ldots, x, \ldots, x(\phi,n))$$

to $P_n$, where the variable $x$ occurs in position $i$ on the right. This completes the construction; we claim that $C_n$ has a solution iff $P_n$ has a semiunifier.

To prove the claim, suppose first that $C_n$ has a solution $\alpha$. Then we may take $R(x)$, the semiunifier, to be $\alpha(x)$ if $x$ is a variable of $C_n$, and $R(x(\phi, j))$ to be $\alpha(y)j$, where $\phi$ is the constraint $x \doteq yi$. Since $\mathcal{R}_n$ is $n$-ary, this is well-defined. Observe that the values of $R$ are rational trees. To check that $R$ is a semiunifier, first notice that the trivial equations of $P_n$ are automatically verified. Now consider an equation of the form

$$y = \sigma(x(\phi,1), \ldots, x, \ldots, x(\phi,n)).$$

We know $\alpha(y)i = \alpha(x)$ because $\alpha$ is a solution in $\mathcal{R}_n$. The desired conclusion thus follows from the identity

$$\alpha(y) = \sigma(\alpha(y)1, \ldots, \alpha(y)i, \ldots, \alpha(y)n).$$

Finally, consider a subsumption constraint $x \leq_i y$. We know that in $\mathcal{R}_n$, $\alpha(x) \sqsubseteq \alpha(y)$. Therefore $\alpha(x) \leq \alpha(y)$ as rational trees, i.e., there is a substitution $T$ mapping $\alpha(x)$ to $\alpha(y)$, and thus $R(x)$ to $R(y)$ as desired.

For the converse, given the semiunifier $R$, we may take the solution $\alpha(x) = R(x)$ for each variable $x$ of the constraint system. It is immediate to check that $\alpha$ is a solution, so this completes the proof.

Now the foregoing lemmas, together with Theorem 1, imply the principal result of this section:

THEOREM 6. *The semiunification problem for rational trees is undecidable.*

REMARK 2. The same method shows that the semiunification problem for arbitrary trees is also undecidable; just apply the "arbitrary feature algebra" version of Theorem 1, and carry out the same proofs. Finally, the method works in the case when all matching substitutions and endomorphisms are required to be one-to-one. This requires checking that the lemmas used in the reductions still hold, and using Theorem 4 instead of Theorem 1.

We close the section with an example of the total construction.

EXAMPLE 4. Consider the feature constraint used in the proof of Theorem 1, derived from the semigroup $Z_2$, the two-element cyclic group. This group is generated by one element $f$, subject to the equation $f^2 = e$. The equation $f = e$ fails in the group ($e$ is the identity.) The equation $f^2 = e$ yields the feature constraint $xf^2 = x$. So for the total feature constraint system we have

$$\begin{array}{rcl} x & \sqsubseteq & z\,; \\ xf & = & z\,; \\ zf & = & x\,; \\ x & \sqsubseteq & y\,; \\ yk & = & a\,; \\ yf & = & w\,; \\ wk & = & b. \end{array}$$

We have expanded the $C_E$ clause to standard form. Converting to a presentation of the semiunification problem for rational trees, we get

$$\begin{array}{rcl} x & \leq_1 & z\,; \\ x & = & \sigma(z, x_1)\,; \\ z & = & \sigma(x, x_2)\,; \\ x & \leq_2 & y\,; \\ y & = & \sigma(t_1, a)\,; \\ y & = & \sigma(w, x_3)\,; \\ w & = & \sigma(t_2, b). \end{array}$$

Our function symbol $\sigma$ is binary since we have two feature symbols $f$ and $k$ in our alphabet. If we draw the finite-state machine for the solution to the feature clause, it is fairly easy to write down the semiunifier $R$. In the example, we have $R(x) = \mu t.\sigma(\sigma(t, x_2), x_1)$, $R(y) = \mu t.\sigma(\sigma(t, b), a)$, $T_1(x_1) = x_2$, $T_2(x_1) = a$, and $T_2(x_2) = b$.

## 6. Conclusion

We think that the setting of the semiunification problem for finite feature algebras is perhaps a more natural way to state the problem than for cyclic terms. Certainly the presentation of the problem is simpler, as is the proof of undecidability. This suggests looking at other unification problems in the feature algebra setting. For example, unification modulo equational theories might have interesting properties.

Our technique does not apply to the original semiunification problem, because it makes essential use of the cyclic property. However, Kfoury, Tiuryn, and Urzyczyn (1990) have shown that the original problem (for path-finite feature algebras) is also undecidable. Their proof uses a very different technique, and one project might be to phrase their results using feature theory. Along this same line, Paris Kanellakis has called our attention to work by Mitchell (1983), and Cosmadakis and Kanellakis (1985), which shows that the inference problem and finite inference problem for mixed functional and inclusion database dependencies are both unsolvable. The techniques used resemble very much the techniques used to solve the two cases of the semiunification problem, though both our reduction and that in (Kfoury et al., 1990) are different from the ones used in these papers. Perhaps feature theory could be used to help understand the full story. We do feel that it is a worthwhile tool for attacking these and similar questions.

## References

Adjan, S. I. (1955). The algorithmic unsolvability of problems concerning certain properties of groups. *Dokl. Akad. Nauk. SSSR*, vol. 103, 533–535 (Russian).

Aït-Kaci, H., and Nasr R. (1986). Logic and Inheritance. In *Proc. 13th ACM Symposium on Principles of Programming Languages*, 219–228.

Aït-Kaci, H., and Podelski, A. (1991). Towards the meaning of LIFE? PRL Research Report 11, Digital Equipment Corporation.

Barwise, Jon (1989). *The situation in logic.* CSLI Lecture Notes 17, Center for Study of Language and Information.

Colmerauer, A. (1988). Prolog and infinite trees. In Clark and Tarnlund, eds. *Logic Programming*, 231–251, Academic Press.

Cosmadakis, S., and Kanellakis, P. (1985). Equational Theories and Database Constraints. In *Proc. 15th ACM Symp. on Theory of Computing*, 273–281.

Dörre, J. (1990). Feature Logic with Weak Subsumption Constraints. IWBS report 101, IBM Deutschland, Stuttgart. (To appear in *Proceedings of 29th Annual Meeting of the ACL*, Berkeley, CA, June 1991.)

Dörre, J., and Rounds, W. (1990). On Subsumption and Semiunification in Feature Algebras. *Proceedings of Fifth IEEE Symposium on Logic in Computer Science*, 300–310.

Henglein, F. (1988). Type inference and semi-unification. In *Proc. 1988 ACM Conference on LISP and Functional Programming*, Snowbird, Utah, 184–197.

Henglein, F. (1989). Polymorphic type inference and semi-unification. Tech. Report 443, New York University.

Hindley, R. (1969). The principal type scheme of an object in combinatory logic. *Transactions of the AMS* 146.

Gurevich, Y. (1966). The word problem for certain classes of semigroups. *Algebra and Logic* vol. 5, 25–35 (Russian). A proof also appears in Gurevich and H. Lewis, The word problem for cancellation semigroups with zero, in *Journal of Symbolic Logic* vol. 49, no.1, 184–191, 1984.

Johnson, Mark (1988). *Attribute-value logic and the theory of grammar.* CSLI Lecture Notes 16, Center for Study of Language and Information.

Kfoury, A. J., Tiuryn, J., and Urzyczyn, P. (1989). Computational consequences and partial solutions of a generalized unification problem. In *4th Annual Symposium on Logic in Computer Science*, 98–105.

Kfoury, A. J., Tiuryn, J., and Urzyczyn, P. (1990). Undecidability of the semiunification problem. In *Proc. 22nd ACM Symposium on Theory of Computing.*

Leiß, Hans (1988). On type inference for object-oriented programming languages. In E. Börger, H. Kleine-Büning, and M. Richter, eds., *CSL '87. 1st Workshop on Computer Science Logic.* Springer LNCS 329, 151–172.

Milner, R. (1978). A theory of type polymorphism in programming. *Jour. Comput. Sys. Sci.* 17, 348–375.

Mitchell, J. C. (1983). The Implication Problem for Functional and Inclusion Dependencies, *Information and Control* 56,3, 154–173.

Moshier, M. (1988). *Extensions to Unification Grammar for the Description of Programming Languages.* Ph.D. thesis, University of Michigan.

Rabin, M.O. (1958). The recursive unsolvability of group-theoretic problems. *Ann. Math.* vol. 67, 172–194.

Rounds, W., and Kasper, R. (1986). A complete logical calculus for record structures representing linguistic information. In *1st Annual Symposium on Logic and Computer Science.*

Schmidt-Schauß, M. (1989). Subsumption in KL-ONE is undecidable. *Proceedings of First International Conference on Principles of Knowledge Representation and Reasoning,* Toronto, 421–431.

Shieber, Stuart (1987). *Introduction to unification-based approaches to grammar.* CSLI Lecture Notes 4, Center for Study of Language and Information.

Shieber, Stuart (1989). *Parsing and type inference for natural and computer languages.* Ph. D. thesis, Stanford University, March 1989. Also appears as SRI Technical Note 460, SRI International.

Slobodskoi, A.M. (1981). Unsolvability of the universal theory of finite groups. *Algebra and Logic* vol. 20, 139–156.

Smolka, Gert (1988). A feature logic with subsorts. LILOG report 33, IBM Deutschland. (To appear, *Journal of Automated Reasoning.*)