Ordering Constraints on Trees

Hubert Comon[1*] and Ralf Treinen[2**]

[1] CNRS and LRI, Bat. 490, Université de Paris Sud, F-91405 ORSAY cedex, France, E-mail: comon@lri.lri.fr

[2] German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, E-mail: treinen@dfki.uni-sb.de

Abstract. We survey recent results about ordering constraints on trees and discuss their applications. Our main interest lies in the family of *recursive path orderings* which enjoy the properties of being total, well-founded and compatible with the tree constructors. The paper includes some new results, in particular the undecidability of the theory of lexicographic path orderings in case of a non-unary signature.

1 Symbolic Constraints

Constraints on trees are becoming popular in automated theorem proving, logic programming and in other fields thanks to their potential to represent large or even infinite sets of formulae in a nice and compact way. More precisely, a *symbolic constraint system*, also called a *constraint system on trees*, consists of a fragment of first-order logic over a set of predicate symbols $\mathcal{P}$ and a set of function symbols $\mathcal{F}$, together with a fixed interpretation of the predicate symbols in the algebra of finite trees $T(\mathcal{F})$ (or sometimes the algebra of infinite trees $I(\mathcal{F})$) over $\mathcal{F}$. The *satisfiability problem* associated with a constraint system is to decide whether a formula has a solution. There are plenty of symbolic constraint systems, some important examples are:

- *unification problems* in which the formulae are conjunctions of equations and where the equality symbol is interpreted as a congruence relation generated by a finite set E of equational axioms. (See [12] for a survey).
- *disunification problems* in which the formulae are conjunctions of equations and negated equations (called *disequations*), or more generally, arbitrary formulae involving no other predicate symbol than equality. Such formulae are interpreted in the free or quotient algebras of $T(\mathcal{F})$. (See [6] for a survey).
- *membership constraints* in which the formulae involve membership constraints of the form $t \in \zeta$ where ζ belongs to an infinite set of *sort expressions*, generally built from a finite set of sort symbols, logical connectives and applications of function symbols. The membership predicate symbols are interpreted using (some kind of) tree automata. (See for example [4]).

* Supported by the Esprit working group CCL, contract EP 6028.

** Supported by the Bundesminister für Forschung und Technology, contract ITW 9105, and by the Esprit working group CCL, contract EP 6028. The result Theorem 8 was obtained while the second author visited LRI.

- *ordering constraints* which are the subject of this survey paper. The set $\mathcal{P}$ now involves, besides equality, a binary predicate symbol $\geq$. This symbol is interpreted as an ordering on trees; we will discuss later which kind of interpretations are relevant.
- many other systems, like *set constraints*, *feature constraints* etc. We refer to [7] for a short survey.

Symbolic constraints, besides their own interest, can be used together with a logical language, hence leading to *constrained formulae.* A constrained formula is a pair (ϕ, c) (actually written $\phi|c$) where ϕ is a formula in some first-order logic built upon a set $\mathcal{Q}$ of predicate symbols and a set $\mathcal{F}'$of function symbols, and c is a formula (called *constraint*) in some constraint system over $\mathcal{P} \subseteq \mathcal{Q}, \mathcal{F} \subseteq \mathcal{F}'$ As sketched above, any constraint system comes with a satisfaction relation $\models$ such that, for any assignment σ of the free variables of c, $\sigma \models c$ iff $c\sigma$ holds in the given interpretation. Then, $\phi|c$ can be simply interpreted as the (possibly infinite) set of formulae

$$[\![\phi \mid c]\!] = \{\phi\sigma \mid \sigma \models c\}$$

It should be clear from the above interpretation that constraints may help in expressing large or infinite sets of formulae. For example, unification problems can be used for compacting the information, allowing for sharing, as in the example:

$$\phi[f(x,x,x)] \quad | \quad x = \text{Bigterm} \quad \text{standing for} \quad \phi[f(\text{Bigterm,Bigterm,Bigterm})]$$

The reader is referred to e.g. [15] for more details.

Constraint systems can also be used in expressing *deduction strategies.* For example, the *basic strategy* for paramodulation and completion can be nicely expressed using the constraint system of unification problems [1, 19]. Let us go further in this direction since this is indeed where ordering constraints come into the picture. First, let us make an excursion into rewrite system theory.

2 Ordered Strategies

Let E be a finite set of equations, for example the classical three equations defining group theory:

$$\begin{cases} (x \times y) \times z = x \times (y \times z) \\ x \times 1 = x \\ x \times x^{-1} = 1 \end{cases}$$

A classical problem is to decide whether a given equation, for example $(x \times y)^{-1} = y^{-1} \times x^{-1}$ in group theory, is a logical consequence of E. This problem, also known as the *word problem*, has been subject to intensive research. The brute force search for a proof using the replacement of equals by equals, although complete, rarely leads to an actual solution. One of the most successful approaches is to use *ordered strategies.* Knuth and Bendix in their famous paper [16] proposed to use the equations in one way only, i.e. as *rewrite rules.* Of course, such a strategy is incomplete in general, but completeness can be restored using a completion mechanism based on the computation of some particular equational consequences called *critical pairs.*

One requirement of the original method was the *termination* of the rewrite system: the replacement of equals by equals using the ordered strategies should always end up after a finite number of replacement steps.

In the above example of group theory, it is quite easy to fulfill this termination requirement by choosing carefully the way in which to orient the equations. The situation changes if we consider the *commutative groups*, adding the equation $x \times y = y \times x$ to the above system. Now the completion procedure fails because commutativity cannot be oriented in either way without loosing termination. Several solutions have been studied to overcome this problem. It is beyond the scope of this paper to investigate all of them (see [10]). They can be mainly divided into two families: rewriting modulo and ordered rewriting. Rewriting modulo seems interesting when the non-orientable axioms are fixed and known, since it is then possible to tailor the computation of critical pairs and any other operation required during the completion process. In general, however, it may also fail. In contrast, ordered completion never fails but may run forever. The idea is very simple: use every equation in one way or the other, depending on the ordering on *the instances on which it is applied.* For example consider the commutativity axiom and assume a total ordering on terms, e.g compare lexicographically the arguments of $\times$, from left to right. Then if $a > b$, $a \times b$ rewrites to $b \times a$ using $x \times y = y \times x$, but not the other way around, since $a \times b > b \times a$, but $b \times a \not> a \times b$. This idea is developed in e.g. [11]. To be more precise, let us introduce some notations.

We use notations consistent with [10]; missing definitions can be found there. A set of *positions* is a (finite) set of strings of positive integers which is closed by prefix and by the lexicographic ordering. Λ is the empty string. For example $\{\Lambda, 1, 2, 21\}$ is a set of position whereas $\{\Lambda, 1, 21\}$ and $\{\Lambda, 2, 21\}$ are not. Given a set of function symbols $\mathcal{F}'$ together with their arity, a *term* t is a mapping from a set of positions P to $\mathcal{F}'$ such that, if $p \in P$ and $t(p)$ has arity n, then $p \cdot n \in P$ and $p \cdot (n+1) \notin P$. $t|_p$ is the subterm of t at position p and $t[u]_p$ is the term obtained by replacing $t|_p$ with u in t (see [10] for the definitions). In $\mathcal{F}'$, we distinguish a particular set of nullary symbols called *variables*. This subset is denoted by $\mathcal{X}$. The set of all positions of a term t is written $Pos(t)$ and the set of its non-variable positions is $\mathcal{F}Pos(t)$.

Now, the deduction rule for the standard completion procedure can be stated as follows:

$$\frac{l \rightarrow r \qquad g \rightarrow d}{l[d]_p\sigma = r\sigma} \quad \text{If } p \in \mathcal{F}Pos(l) \text{ and } \sigma = \mathrm{mgu}(l|_p, g)$$

This rule is classically associated with an *orientation rule* w.r.t. a given ordering on terms:

$$\frac{l = r}{l \rightarrow r} \quad \text{If } l > r$$

Now the ordered completion consists of a single rule (besides simplification rules which we do not consider so far):

$$\frac{l = r \qquad g = d}{l[d]_p\sigma = r\sigma} \quad \text{If } p \in \mathcal{F}Pos(l),\ \sigma = mgu(l|_p, g),\ l\sigma \not\leq r\sigma \text{ and } g\sigma \not\leq d\sigma$$

which deduces a new equation only for equations which actually can form a critical pair.

In the light of constrained logics, this rule can be reformulated as the (classical) critical pair computation between $l = r \mid l \not\leq r$ and $g = d \mid g \not\leq d$. Going further in this direction it is possible to improve the above deduction rule, expressing the conditions at the object level, thus keeping track of which instances of the equations can lead to a critical pair. We get then the following constrained deduction rule:

$$\frac{l = r \mid c \qquad g = d \mid c'}{l[d]_p = r \ \mid \ l|_p = g \wedge c \wedge c' \wedge l > r \wedge g > d} \quad \text{If } p \in \mathcal{F}Pos(l)$$

(Note that we replaced here $\not\leq$ by $>$, assuming that the ordering is total on ground terms). This strategy is strictly more restrictive than the ordered deduction rule because we keep track of the reason why some former equations have been generated: the constraint contains in some sense the "history" of the deduction. This point of view has been extended to arbitrary clauses and shown to be complete (see e.g. [20]).

This new rewriting point of view has however a drawback: at some point it is necessary to decide whether the constraint is indeed satisfiable: all these systems are quite useless if we are computing with empty sets $[\![\phi \mid c]\!]$. This is the motivation for the study of *ordering constraint solving* which is the subject of the next sections. First we will precise which interpretations of the ordering are relevant.

3 Orderings on Trees

With respect to ordered strategies in first-order logic with equality, the ordering we consider must have the following properties:

- To be well founded
- To be *monotonic* i.e. $f(\ldots, s, \ldots) > f(\ldots, t, \ldots)$ whenever $s > t$.
- To be total on ground terms. (i.e. terms without variables).

Totality is mandatory only for completeness of the strategy, whereas the two first properties are already necessary for the completeness of the rules themselves. Monotonicity is required because, along the proofs, equality steps can take place at any positions in the terms.

Typical orderings which fulfill the above three properties are the *recursive path orderings* introduced by N. Dershowitz [9]. We consider these orderings as well as some extensions in sections 4, 5.

Originating from quite different problems, other interpretations of the orderings have been studied in the literature. For example, $\geq$ can be interpreted as the *subterm ordering*. To be more precise, let us introduce some terminology. The *existential fragment* of a the theory of $\mathcal{P}, \mathcal{F}$ (in a given interpretation) is the set of formulae $\exists \mathbf{x}.\phi$ which hold in the interpretation, where ϕ is any quantifier-free formula built over $\mathcal{P}, \mathcal{F}$ and $\mathbf{x}$ is the set of variables occurring in ϕ. More generally, the Σ_n fragment of the theory is the set of (closed, i.e. without free variables) formulae $\exists^* \mathbf{x_1} \forall^* \mathbf{x_2} \exists^* \ldots \mathbf{x_n}.\phi$ which hold true in the interpretation, where ϕ is quantifier free. It is shown in [26] that existential fragment of the theory of subterm ordering is decidable. On the other side, it is also shown in [26] that the Σ_2 fragment of the theory of subterm ordering is undecidable, which sets up a quite precise boundary between decidability and undecidability in this case. Subterm ordering is also studied in the case of infinite trees: again the existential fragment of the theory is decidable [25] and the Σ_2 fragment is undecidable [24].

Let us finally consider yet another ordering on trees: the *encompassment ordering*. We say that s encompasses t (noted $s \trianglerighteq t$) if some instance of t is a subterm of s. For example, $s = g(f(f(a,b), f(a,b)))$ encompasses $t = f(x,x)$ since instantiating x with $f(a,b)$, we get a term $t\sigma$ which is a subterm of s. The encompassment ordering plays a central role in the so-called *ground reducibility problem* in rewriting theory. Given a rewrite system $\mathcal{R}$, a term t is ground reducible w.r.t. $\mathcal{R}$ if all the ground instances of t (i.e. instances without variables) are reducible by $\mathcal{R}$. A reducible term is always ground reducible, but the converse is false. For example, consider $\mathcal{R} = \{s(s(0)) \to 0\}$ and $t = s(s(x))$ and assume that the set of function symbols only consists of $0, s$. Then t is ground reducible because the tail of any of its ground instances will be $s(s(0))$. However, it is not reducible. Ground reducibility has been shown decidable by D. Plaisted [22]. However, as noticed in [3], this property can be nicely expressed using the encompassment ordering: t is ground reducible by a rewrite system whose left members are $l_1, \ldots, l_n$ iff

$$\forall x, \mathbf{z}.\ x \trianglerighteq t \to (x \trianglerighteq l_1 \vee \ldots \vee x \trianglerighteq l_n)$$

where $\mathbf{z}$ is the set of variables of t.

Theorem 1 [3]. *The first-order theory of finitely many (unary) predicate symbols* $\trianglerighteq l_1, \ldots, \trianglerighteq l_n$ *is decidable.*

This shows in particular that ground reducibility is decidable.

4 Recursive Path Ordering Constraints

4.1 The lexicographic path ordering

Given a *precedence* $\geq_{\mathcal{F}}$ (which we assume so far to be an ordering) on $\mathcal{F}$, the *lexicographic path ordering* on $T(\mathcal{F})$ is defined as follows: $s = f(s_1, \ldots, s_n) >_{lpo} g(t_1, \ldots, t_m) = t$ iff one of the following holds:

- $f >_{\mathcal{F}} g$ and, for all i, $s >_{lpo} t_i$
- for some i, $s_i \geq_{lpo} t$
- $f = g$ (and $n = m$) and there is a $j < n$ such that
  - $s_1 = t_1, \ldots, s_j = t_j$ and $s_{j+1} >_{lpo} t_{j+1}$
  - and, for all i, $s >_{lpo} t_i$

Proposition 2 [9, 14]. $\geq_{lpo}$ *is a well-founded ordering. It is monotonic and, if* $\geq_{\mathcal{F}}$ *is total on* $\mathcal{F}$*, then* $\geq_{lpo}$ *is total on* $T(\mathcal{F})$.

This shows, according to the previous section, that the lexicographic path ordering is a good candidate for ordered strategies. Fortunately, there is a positive result on constraint solving in this interpretation:

Theorem 3 [5]. *The existential fragment of the theory of a total lexicographic path ordering is decidable.*

The original proof has been actually simplified in [18] where two other problems are considered: the satisfiability *over an extended signature* and complexity issues. A conjunction of inequations, built over an initial set of function symbols $\mathcal{F}$ is satisfiable over an extended signature if there is an (finite) extension $\mathcal{F} \cup \mathcal{F}'$ of the set of function symbols and an extension of the precedence to this new set of function symbols in which the formula is satisfiable. This kind of interpretation is actually useful for the applications in automated theorem proving (see [20]).

Theorem 4 [18]. *The satisfiability problems for quantifier-free total LPO ordering constraints over a given signature and over an extended signature are both NP-complete.*

Actually, the NP-hardness result can be strengthened:

Proposition 5. *Let $\geq$ be interpreted as a total $\geq_{lpo}$. Deciding satisfiability of a single inequation $s > t$ is NP-complete.*

Sketch of the proof. According to the above theorem, we only have to prove NP-hardness. We encode 3SAT. $\mathcal{F} = \{f, g, h, 0\}$ with the precedence $g > h > f > 0$ and we assume g unary, h, f binary and 0 constant. We will use also the abbreviations: $1 = f(0,0)$ and $2 = f(0, f(0,0))$. Then, we use the following translations:

- each positive literal P is translated into $h(2, x_P) > f(h(x_P, x_P), h(2,0))$ which holds iff x_P is assigned to 1.
- each negative literal $\neg P$ is translated into $1 > x_P$ which holds iff x_P is assigned to 0.
- each clause $s_1 > t_1 \;\vee\; s_2 > t_2 \;\vee\; s_3 > t_3$ is equivalent (w.r.t. the $\geq_{lpo}$ interpretation) to

$$f(g(C_1(C(0))), f(g(C_2(C(0))), g(C_3(C(0))))) > h(0, g(C(C(0))))$$

  where $C(x) \stackrel{\text{def}}{=} f(t_1, f(t_2, f(t_3, x)))$, $C_1(x) \stackrel{\text{def}}{=} f(s_1, f(t_2, f(t_3, x)))$, $C_2(x) \stackrel{\text{def}}{=} f(t_1, f(s_2, f(t_3, x)))$ and $C_3(x) \stackrel{\text{def}}{=} f(t_1, f(t_2, f(s_3, x)))$.
- the conjunction $s_1 > t_1 \wedge \ldots \wedge s_n > t_n$ is equivalent to the single inequation

$$\begin{aligned} &C_h(s_1, \ldots, s_n, t_1, \ldots, t_n) \\ &\quad > C_f(C_h(t_1, s_2, \ldots, s_n, t_1, \ldots, t_n), \ldots, C_h(s_1, \ldots, s_{n-1}, t_n, t_1, \ldots, t_n)) \end{aligned}$$

  where C_h and C_f are the right "combs" recursively defined by: $C_\alpha(t, L) \stackrel{\text{def}}{=} \alpha(t, C_\alpha(L))$ and $C_\alpha(\emptyset) \stackrel{\text{def}}{=} 0$.

The coding is in $O(n^2)$. It is a routine verification that the resulting inequation is satisfiable iff the set of clauses is satisfiable. □

The proposition also holds for satisfiability over an extended signature, with a minor modification: $\neg P$ has to be translated in a slightly more complicated way: $f(0, f(1, x_P)) > f(1,0) \wedge f(0, f(1,0)) > f(1, x_P)$ which is in turn expressed using a single inequation as we did above.

4.2 The recursive path ordering with status

The recursive path ordering with status is slightly more general than the lexicographic path ordering. In addition to the precedence, we assume, for each function symbol, given a *status* which can be either "multiset" or "lexicographic" (other status are also available, but w.r.t. constraint solving only these two are relevant).

The definition of the ordering is exactly the same as in section 4.1 except when $f = g$. In that case, we get the status of f and compare the terms as before if the status is lexicographic, whereas, if the status is multiset, $s >_{rpo} t$ iff $\{s_1, \ldots, s_n\} \gg \{t_1, \ldots, t_n\}$ where $\gg$ is the multiset extension of $>_{rpo}$ (see [9, 10] for more details). This ordering is not total on ground terms as permuting the direct subterms of a function symbol whose status is multiset leads to incomparable terms. However, modulo such permutations, the (quasi-)ordering is total. With such an extension to a total quasi-ordering, constraint solving is still possible:

Theorem 6 [13]. *The existential fragment of the theory of a total recursive path (quasi-)ordering with status is decidable.*

Actually, as above, the fragment is NP-complete. Satisfiability over an extended signature is NP-complete as well [18].

4.3 Partial recursive path orderings

Although less interesting from the applications point of view, the question arises of whether the above results can be extended to arbitrary (non-total) recursive path orderings. This turns out to be a difficult question, which is not answered so far.

The only progress in this direction is the study of *tree embedding constraints*. This is yet another interpretation of the ordering on trees. Tree embedding is the least recursive path ordering: it extends the precedence where any two symbols are uncomparable. It can also be defined as the least monotonic ordering which contains the subterm relation. Up to our knowledge, there is only one result about tree embedding and, more generally, partial recursive path orderings:

Theorem 7 [2]. *The positive existential fragment of the theory of tree embedding is decidable.*

In the *positive* existential fragment, negation is not allowed in the quantifier-free part of the formula.

4.4 The first-order theory of recursive path orderings

Now, extending the language allowing for some more quantifiers may be useful for deciding some other properties (such as for *simplification rules* as described in [15]). Unfortunately, we fall into the undecidability side as soon as we try to enlarge the class of formulae. R. Treinen first shows that the Σ_4 fragment of the theory of a partial lexicographic path ordering is undecidable [24]. But this leaves still some room and most properties for which a decision procedure would be welcome can be expressed in the Σ_2 fragment. Moreover the result did not apply to total orderings, which are the most interesting ones. Extending the technique of [24], it is possible to show the following:

Theorem 8. *The Σ_2 fragment of the theory of any (partial or total) lexicographic path ordering is undecidable, as soon as there is at least a binary function symbol.*

We give a sketch of the proof, the full (quite technical) proof of this result can be found in [8].

We reduce the Post Correspondence Problem (PCP) to the theory of a lexicographic path ordering following the line of [24]. Let F be a finite set of function symbols, such that 0 is a minimal constant, f is a binary function symbol which is minimal in $F-\{0\}$ and g is a minimal unary symbol larger than f. Let $P=(p_i,q_i)_{i=1\ldots n}$ be an instance of the PCP over the alphabet $\{a,b\}$. We can device an injective coding function $\mathsf{cw}\colon \{a,b\}^* \to T(\{f,0\})$ and formulae $\underline{\text{empty}}(x)$ and $\underline{\text{prefix}}_v(x,y)$ for every $v\in\{a,b\}^*$, such that $\models \underline{\text{empty}}(x)$ iff $x=\mathsf{cw}(\epsilon)$, and that $\models \underline{\text{prefix}}_v(x,\mathsf{cw}(w))$ iff $x=cw(v\circ w)$. Now it is not hard to device an injective pairing function $\mathsf{pair}\colon T(\{f,0\})\times T(\{f,0\}) \to T(\{f,0\})$ and a formula $x \sqsupset y$,such that

$$\mathsf{pair}(x,y) \sqsupset \mathsf{pair}(x',y') \leftrightarrow \bigvee_{(p,q)\in P} \underline{\text{prefix}}_p(x,x') \wedge \underline{\text{prefix}}_q(y,y')$$

and such that $\sqsupset$ is well-founded but nevertheless $t \sqsupset t'$ implies $t <_{lpo} t'$. Intuitively, $t \sqsupset t'$ reads "the pair represented by t' is obtained form the pair represented by t by one construction step of P. It is important that $\sqsupset$ is a well-founded relation, this can be achieved by counting in $\sqsupset$ (not in pair) the maximal number of construction steps to go.

The idea is now to design a sentence $\underline{\text{solv}}$ which holds iff there is a sequence $t_0,\ldots,t_n$ such that $t_0=\mathsf{pair}(\mathsf{cw}(\epsilon),\mathsf{cw}(\epsilon))$, $t_n=\mathsf{pair}(\mathsf{cw}(w),\mathsf{cw}(w))$ for some $w\neq\epsilon\in\{a,b\}^*$ and $\models t_i \sqsupset t_{i+1}$ for every $0\leq i<n$. Let $\mathcal{I}(x)$ be a formula which holds iff $x=\mathsf{pair}(\mathsf{cw}(\epsilon),\mathsf{cw}(\epsilon))$, and let $\mathcal{F}(x)$ be a formula which holds iff $x=\mathsf{pair}(\mathsf{cw}(w),\mathsf{cw}(w))$ for some $w\neq\epsilon\in\{a,b\}^*$. In the following formula $\underline{\text{solv}}$, some parts are not yet defined. The intended meaning of $x\,\underline{\text{head}}\,y$ is that x is the head of the sequence y, $\underline{\text{nonempty}}(y)$ expresses that y has a head and $(x,y')\,\underline{\text{sub}}\,y$ should express that the sequence $cons(x,y')$ is a subsequence of y.

$$\begin{aligned}&\exists y(\ \exists x,y'(\mathcal{I}(x)\wedge (x,y')\,\underline{\text{sub}}\,y)\wedge\\ &\quad \forall x,y'((x,y')\,\underline{\text{sub}}\,y \to [\mathcal{F}(x)\vee(\underline{\text{nonempty}}(y')\wedge \forall x'(x'\,\underline{\text{head}}\,y' \to x\sqsupset x'))]\end{aligned}$$

Now, we have to show that the above formula $\underline{\text{solv}}$ holds iff P has a solution. We give first some characterizations of the "if" and "only if" parts respectively in terms of properties of the formulas $\underline{\text{nonempty}}$(x), x $\underline{\text{head}}$ y and (x,y) $\underline{\text{sub}}$ z. Then, we will sketch how (x,y) $\underline{\text{sub}}$ z is constructed. This is the most complicated part; the constructions of x $\underline{\text{head}}$ y and $\underline{\text{nonempty}}$(x) are skipped here. We will also sketch why (x,y) $\underline{\text{sub}}$ z follows the requirements.

In order to show that $\underline{\text{solv}}$ holds if P has a solution, we have to design a coding cs of sequences of elements from $T(\{f,0\})$. $\mathsf{cs}\colon T(\{f,0\})^*\to T(F)$. This is given by $\mathsf{cs}(\epsilon)\overset{\text{def}}{=}0$, and $\mathsf{cs}(cons(t,\bar{t}))\overset{\text{def}}{=} f(g(t),\mathsf{cs}(\bar{t}))$. Now, $\underline{\text{solv}}$ holds if P has a solution, provided that the following relations are satisfied:

$$\begin{aligned}&\models \underline{\text{nonempty}}(\mathsf{cs}(s)) &&\Leftrightarrow s\neq\epsilon\\ &\models t\,\underline{\text{head}}\,\mathsf{cs}(t_0,\ldots,t_n) &&\Leftrightarrow t_0=t\\ &\models (t,u')\,\underline{\text{sub}}\,\mathsf{cs}(t_0,\ldots,t_n) &&\Leftrightarrow \text{exists } i\leq n,\ t=t_i,\ u'=\mathsf{cs}(t_{i+1},\ldots,t_n)\end{aligned}\tag{1}$$

Once we have the definition of $\underline{\text{sub}}$ with property (1), it follows immediately that $\underline{\text{solv}}$ holds if P has a solution: We take y to be the coding of the solution to P.

Conversely, P has a solution if $\underline{\text{solv}}$ holds, provided that the following relations are satisfied:

$$\underline{\text{nonempty}}(y) \rightarrow \exists x\, x\, \underline{\text{head}}\, y \tag{2}$$

$$(x, y')\, \underline{\text{sub}}\, y \wedge x'\, \underline{\text{head}}\, y' \wedge x \sqsupset x' \rightarrow \exists y''\, (x', y'')\, \underline{\text{sub}}\, y \tag{3}$$

This claim is easily proven by well-founded induction on $\sqsupset$. The lemmata (2) and (3) give exactly the argument needed in the induction step. Using well-founded induction at this place is a central idea in [24].

Appropriate definitions of $\underline{\text{nonempty}}(y)$ and $x\, \underline{\text{head}}\, y$ are given easily. Now, let us sketch the construction of $(x, y)\, \underline{\text{sub}}\, z$. The first step is the definition

$$\phi_1(x, y) \stackrel{\text{def}}{=} f(g(x), g(x)) \geq y > g(x)$$

It is easily proven by structural induction on u, that $\models \phi_1(t, u)$ implies that $g(t)$ is the maximal subterm of u which is headed by a symbol not smaller than g. For instance, if g is the greatest symbol in F, this means that $g(t)$ is the maximal g-headed subterm of u. In this proof, we exploit the fact that $f < g$. It is not always true, that for any y containing a g there is an x such that $\phi_1(x, y)$. On the other hand, the definition of $\underline{\text{nonempty}}(y)$ will have to ensure this fact, as can be seen from the definition of $\underline{\text{sub}}$ given below. The formula $\exists x\, \phi_1(x, y)$ does the job but introduces an existential quantifier at the wrong place, which would throw $\underline{\text{solv}}$ out of the Σ_2 fragment. A working formula $\psi(y)$ using only universal quantifiers can be found in the full paper [8]. Now it can be shown that always

$$\models \phi_1(x, \mathsf{cs}(t_0, \ldots, t_n)) \leftrightarrow x = t_n \tag{4}$$

which gives us access to the greatest pair of a list. Note that in our representation of lists, the greatest term stands at an *innermost* position; it is by no means obvious that we can access this term when the ordering might be total. This was a main difficulty which was not solved in the result on partial precedences in [24]. The complete definition of $(x, y')\, \underline{\text{sub}}\, y$ is

$$\begin{aligned}&\big(\phi_1(x, y) \wedge y' = 0\big)\\ &\vee \exists w\, \big(f(g(x), f(g(x), y')) > y \geq f(g(x), y') > g(w) > g(x) \wedge \phi_1(w, y)\big)\end{aligned}$$

Let us sketch now the main part of the proof, namely that the definition of $(x, y')\, \underline{\text{sub}}\, y$ satisfies (1). The "$\Leftarrow$" direction of (1) is easy, let us prove the "$\Rightarrow$" direction. If the first case of $\underline{\text{sub}}$ applies, then the claim holds by (4). Otherwise,

$$\begin{aligned}&\models f(g(t), f(g(t), u')) > \mathsf{cs}(t_0, \ldots, t_n) \geq f(g(t), u') > g(r) > g(t)\\ &\quad \wedge \phi_1(r, \mathsf{cs}(t_0, \ldots, t_n))\end{aligned}$$

holds for some $r \in T(F)$. In fact, $r = t_n$ by (4). Now, $\models g(r) > g(t)$, hence $t_n >_{lpo} t$. Let i be the smallest index such that $t_i \geq_{lpo} t$. Such an i exists since $t_n >_{lpo} t$. Hence, $t_{i'} \not\geq_{lpo} t$ for all $i' < i$. Using the lpo rules, $\mathsf{cs}(t_0, \ldots, t_n) \geq_{lpo} f(g(t), u')$ is simplified into $\mathsf{cs}(t_i, \ldots, t_n) \geq_{lpo} f(g(t), u')$, hence $\mathsf{cs}(t_i, \ldots, t_n) >_{lpo} u'$.

Now let j be the smallest index such that $t \not\geq_{lpo} t_j$. Note that j is well defined since $t \not\geq_{lpo} t_n$. Since $f(g(t), f(g(t), u')) >_{lpo} \mathsf{cs}(t_0, \ldots, t_n)$, it follows that $f(g(t), f(g(t), u')) >_{lpo} \mathsf{cs}(t_j, \ldots, t_n)$. Since by construction $t \not\geq_{lpo} t_j$, this inequality is equivalent to $u' \geq_{lpo} \mathsf{cs}(t_j, \ldots, t_n)$. Together we have

$$\mathsf{cs}(t_i, \ldots, t_n) >_{lpo} u' \geq_{lpo} \mathsf{cs}(t_j, \ldots, t_n)$$

and hence $i < j$. By our construction of j this means $t \geq_{lpo} t_i$. On the other hand we have $t_i \geq_{lpo} t$, hence $t = t_i$. Using the definition of an lpo, we can now simplify

$$\begin{aligned} f(g(t_i), f(g(t_i), u')) >_{lpo} \mathsf{cs}(t_0, \ldots, t_n) &\Rightarrow^* f(g(t_i), f(g(t_i), u')) >_{lpo} \mathsf{cs}(t_i, \ldots, t_n) \\ &\Rightarrow\ f(g(t_i), u') >_{lpo} \mathsf{cs}(t_{i+1}, \ldots, t_n) \\ &\Rightarrow\ u' \geq_{lpo} \mathsf{cs}(t_{i+1}, \ldots, t_n) \end{aligned}$$

On the other hand, we have

$$\begin{aligned} \mathsf{cs}(t_0, \ldots, t_n) \geq_{lpo} f(g(t_i), u') &\Rightarrow^* \mathsf{cs}(t_i, \ldots, t_n) \geq_{lpo} f(g(t_i), u') \\ &\Rightarrow\ \mathsf{cs}(t_{i+1}, \ldots, t_n) \geq_{lpo} u' \end{aligned}$$

Hence, $u' = \mathsf{cs}(t_{i+1}, \ldots, t_n)$. $\square$

In case there are only unary symbols we can use another reduction technique and show:

Proposition 9. *The first-order theory of strings embedding is undecidable.*

The theory of strings involves a binary concatenation function, but the undecidability result in fact holds if we restrict ourselves to unary functions which prefix a string with a fixed symbol. With the representation of strings as terms, this kind of left concatenation corresponds to the application of a unary function symbol.

Sketch of the proof: We encode the concatenation of words, whose first-order theory is known to be undecidable (see e.g. [23]). We use an additional symbol # and successively express the following properties:

$x\# \leq z$, where x contains no #:

$$\phi_1(x, z) \stackrel{\text{def}}{=} x \leq z \wedge \forall y(\#y \leq z \leftrightarrow y = \Lambda)$$

$z = x\#y$ (and x, y are #-free):

$$\begin{aligned} \phi_2(x, y, z) \stackrel{\text{def}}{=}\ & \# \not\leq x \wedge \# \not\leq y \wedge \#\# \not\leq z \wedge \\ & \forall u[\#\# \not\leq u \rightarrow (z \leq u \leftrightarrow (\phi_1(x, u) \wedge \#y \leq u))] \end{aligned}$$

This reads: "z is minimal with the property that z contains at most one #, $x\# \leq z$ and $\#y \leq z$."

x, y, u are #-free and $z = xy$:

$$\phi_3(x, y, u) \stackrel{\text{def}}{=} \exists z.(\phi_2(x, y, z) \wedge \# \not\leq u \wedge \forall v(u \leq v \leq z \leftrightarrow (v = u \vee v = z)))$$

Since u doesn't contain #, it must be the immediate predecessor of z obtained by deleting the # of z. $\square$

The decidability of the theory of a total lexicographic path ordering on strings remains open.

5 Extensions

We list below a number of extensions which have still to be investigated.

- As we have seen in section 2, using ordering constraints avoids failure even in presence of associative-commutative (AC) function symbols. This particular case of unorientable equations occurs very often. On the other hand, however, although the use of ordering constraints prevents failure, completion procedures often run forever in such situations. Hence, from the practical point of view, it is important to design dedicated techniques for this particular situation. In general, AC equations are not treated like the other relations; this theory is built-in, which implies the use of AC-unification (or AC equality constraints). Using ordering constraints in this context requires first an AC-compatible ordering which is total on ground terms. For a long time no such ordering was known. P. Narendran and M. Rusinowitch [17] were the first to give such an ordering, which is based on polynomial interpretations. An rpo-style AC-compatible ordering, total on ground terms was then given in [21]. Is it possible to design a constraint solving algorithm for such an ordering? This is an open question which is currently under investigation.
- Another important question is the *combination* of constraint systems on terms. Indeed, we may consider the problem of using ordered strategies on constrained equations (or clauses). The combination of ordering constraints and equations and disunification constraints is quite obvious (equational constraints are already considered within the ordering constrains and $s \neq t$ is equivalent to $s > t \vee t > s$ when the ordering is total). More relevant is the combination with membership constraints. This is another open question currently under investigation: is the existential fragment of the theory of $\geq, \in \zeta$, for a family of unary predicate symbols $\in \zeta$, as explained in introduction, decidable?
- Finally, we already mentioned some open questions about the theory of recursive path orderings. In case of partial orderings, we don't know whether the existential fragment is decidable. Similarly, the problem of the first-order theory of a total lexicographic path ordering on unary function symbols is open.

References

1. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In D. Kapur, editor, *Proc. 11th Int. Conf. on Automated Deduction, Saratoga Springs, NY*, Lecture Notes in Computer Science, vol. 607, pages 462–476. Springer-Verlag, June 1992.
2. A. Boudet and H. Comon. About the theory of tree embedding. In M. C. Gaudel and J.-P. Jouannaud, editors, *Proc. Int. Joint Conf. on Theory and Practice of Software Development*, Lecture Notes in Computer Science, vol. 668, pages 376–390, Orsay, France, Apr. 1993. Springer-Verlag.
3. A.-C. Caron, J.-L. Coquidé, and M. Dauchet. Encompassment properties and automata with constraints. In C. Kirchner, editor, *Proc. 5th. Int. Conf. on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, vol. 690, pages 328–342, Montreal, Canada, 1993. Springer-Verlag.

4. H. Comon. Equational formulas in order-sorted algebras. In *Proc. 17th Int. Coll. on Automata, Languages and Programming, Warwick*, Lecture Notes in Computer Science, vol. 443, pages 674–688, Warwick, July 1990. Springer-Verlag.
5. H. Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.
6. H. Comon. Disunification: a survey. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 322–359. MIT Press, 1991.
7. H. Comon. Constraints in term algebras (short survey). In T. R. M. Nivat, C. Rattray and G. Scollo, editors, *Proc. Conf. on Algebraic Methodology and Software Technology*, Univ. of Twente, 1993. Springer Verlag, series Workshop in Computing. Invited talk.
8. H. Comon and R. Treinen. The first-order theory of lexicographic path orderings is undecidable. Research Report RR-93-42, Deutsches Forschungszentrum für Künstliche Intelligenz, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany, Sept. 1993. Anonymous ftp from `duck.dfki.uni-sb.de:/pub/ccl/dfki-saarbruecken`.
9. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, Mar. 1982.
10. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.
11. J. Hsiang and M. Rusinowitch. On word problems in equational theories. In *Proc. in 14th ICALP Karlsruhe*, July 1987.
12. J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 257–321. MIT-Press, 1991.
13. J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. In *Proc. 18th Int. Coll. on Automata, Languages and Programming, Madrid*, Lecture Notes in Computer Science, vol. 510, pages 455–468, 1991. Springer-Verlag.
14. S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. Available as a report of the department of computer science, University of Illinois at Urbana-Champaign, 1980.
15. C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automatic deduction.
16. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.
17. P. Narendran and M. Rusinowitch. Any ground associative-commutative theory has a finite canonical system. In R. V. Book, editor, *Proc. 4th Rewriting Techniques and Applications, Como*, Lecture Notes in Computer Science, vol. 488, pages 423–434. Springer-Verlag, Apr. 1991.
18. R. Nieuwenhuis. Simple LPO constraint solving methods. *Inf. Process. Lett.*, 47:65–69, Aug. 1993.
19. R. Nieuwenhuis and A. Rubio. Basic superposition is complete. In B. Krieg-Bruckner, editor, *Proc. European Symp. on Programming*, Lecture Notes in ComputerScience, vol. 582, pages 371–389, Rennes, 1992. Springer-Verlag.
20. R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In D. Kapur, editor, *Proc. 11th Int. Conf. on Automated Deduction, Saratoga Springs, NY*, Lecture Notes in Computer Science, vol. 607, pages 477–491. Springer-Verlag, June 1992.

21. R. Nieuwenhuis and A. Rubio. A precedence-based total AC-compatible ordering. In C. Kirchner, editor, *Proc. 5th Rewriting Techniques and Applications, Montréal*, Lecture Notes in Computer Science, vol. 690, pages 374–388. Springer-Verlag, June 1993.
22. D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.
23. W. V. Quine. Concatenation as a basis for arithmetic. *Journal of Symbolic Logic*, 11(4), 1946.
24. R. Treinen. A new method for undecidability proofs of first order theories. *Journal of Symbolic Computation*, 14(5):437–458, Nov. 1992.
25. S. Tulipani. Decidability of the existential theory of infinite terms with subterm relation. To appear in *Information and Computation*, 1994.
26. K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebras. *J. ACM*, 34(2):492–510, 1987.