Inductive proofs by specification transformations

Hubert COMON*

Abstract

We show how to transform equational specifications with relations between constructors (or without constructors) into order-sorted equational specifications where every function symbol is either a free constructor or a completely defined function.

This method allows to reduce the problem of inductive proofs in equational theories to Huet and Hullot's proofs by consistency [HH82]. In particular, it is no longer necessary to use the so-called "inductive reducibility test" which is the most expensive part of the Jouannaud and Kounalis algorithm [JK86].

Introduction

Let F be a set of function symbols together with their profile (for example, $F = \{0 :\to int2;\ succ : int2 \to int2\}$) and E be a finite set of equational axioms (for example $E = \{succ(succ(0)) = 0\}$). The problem of inductive proofs in equational theories is to decide whether an equation (whose variables are implicitly universally quantified) is valid in $T(F)/=_E$, the quotient algebra of the terms constructed on F by the congruence generated by E. (For example, $succ(succ(x)) = x$ is an inductive theorem in the specification (F, E) but is not an equational consequence of E).

The "proof by consistency" method [KM87] consists in adding to E the theorem to be proved and trying to deduce a contradiction (inconsistency) using equational reasoning. This method has been widely studied. Let us cite among others [Mus80,Gog80,Lan81,HH82,KM87], [Kuc87,JK86,Fri86,KNZ86,Bac88].

All these works use the Knuth-Bendix completion procedure as a basis for equational deduction: E is assumed to be oriented into a ground convergent term rewriting system $\mathcal{R}$. If the completion procedure constructs for $\mathcal{R} \cup \{s = t\}$ (where $s = t$ is the theorem to be proved) a (ground) convergent term rewriting system without deriving an inconsistency, then $s = t$ is an inductive theorem[1]. If an inconsistency is derived, $s = t$ is not an inductive theorem.

The papers cited above essentially differ in the assumptions they make on F, E and in the way they detect inconsistencies. For example, in Musser's paper [Mus80] E is assumed to contain a complete axiomatization of an equality predicate and an inconsistency is derived simply when the completion procedure generates the equation $true = false$.

In Huet and Hullot's method [HH82], F is assumed to be split into two disjoint sets C (constructors) and D (defined operators) with the following conditions :

*Laboratoire d'Informatique fondamentale et d'Intelligence Artificielle, Institut IMAG, 46 Ave. Félix Viallet, 38031 Grenoble cedex, France. E-mail : comon@lifia.imag.fr

[1] In [Bac88] the requirement for the resulting term rewriting system to be convergent has been weakened.

- every term constructed on C only is irreducible by $\mathcal{R}$
- every term in $T(F) - T(C)$ is reducible by $\mathcal{R}$

Then, an inconsistency is detected when the completion procedure generates an equation $s = t$ between two "constructor terms" (i.e built without any symbol of D).

This method was generalized in the so-called "inductive completion procedure" by Jouannaud and Kounalis [JK86] where the requirement on F to be split into constructors and defined operators is dropped. They show that the key concept for detecting inconsistencies is the "inductive reducibility test". A term is said to be inductively reducible when all its ground instances are reducible. For example, $succ(succ(x))$ is inductively reducible by $succ(succ(0)) \rightarrow 0$ (but it is not reducible). Then, an inconsistency is detected when the completion procedure generates an equation $s = t$ where $s > t$ (for a given simplification ordering containing $\rightarrow_{\mathcal{R}}$) and s is not inductively reducible.

Recently Bachmair [Bac88], refining the equational consequences to be added during the completion procedure, proved that it is not necessary to orient the equations computed by the inductive completion procedure. In this case, an inconsistency is detected when a non inductively reducible *equation* is derived.

Although inductive reducibility has been shown to be decidable [Pla85,Com88a], Plaisted's algorithm as well as others ([KNZ85,KNZ86,JK86] for example) are very complex. Actually, they are at least twice exponential and (except for [Com88a]) cannot be used in practice.

The aim of this paper is to show that it is possible to reduce the general case handled in [JK86,Bac88] to Huet and Hullot's method by transforming the specification. This allows to avoid the inductive reducibility test since such a test is trivial in Huet and Hullot's algorithm.

Given a Term Rewriting System (TRS for short) $\mathcal{R}$, it is shown in [CR87,Com88b,Com88a] how to compute a conditional grammar for the set NF of the ground terms which are irreducible by $\mathcal{R}$. This construction is performed using equational problems simplification [CL88].

In [Com88b,Com88a] a cleaning algorithm for conditional grammars is given. This provides a method for deciding inductive reducibility in the general case but can also be used for computing an order-sorted specification which is equivalent (in some suitable sense) to the original specification and where F is split into constructors and defined operators. Such a construction can be extended in order to handle order-sorted specifications as well.

Another specification transformation was already proposed in [Tha85] in a very specific case. This paper shows that, whenever there are no overlap between left hand sides of the rules, when the set of function symbols is split into constructors and defined functions, and when the TRS is left linear, then the signature can be enriched with new constructor symbols in order to have the additional property that no rule contains "inner" occurrences of a defined symbol. Such a transformation is similar to ours since we actually add some new constructors. However we don't make the above mentioned assumptions and give very different (stronger) results.

Also, Kapur and Musser [KM86] proposed some specification transformations related to proofs by consistency. However, they do not address the same problem. Roughly, they assume some information about "what should be" the initial algebra (i.e. what should be the constructors) and then complete the set of axioms (in some consistent way) in order to indeed get this initial algebra. At the opposite, we want to preserve the initial algebra (up to isomorphism) and we allow some relations between constructors (or, more generally, we do not assume that a set of constructors has been defined at all). Then we show in this paper how to compute free generators of the initial algebra.

We present the transformation in section 2 and state the basic properties of the resulting specification. Theorem 4 is the main (new) result of the paper. Then we show in section 3 how to perform inductive proofs in the resulting order-sorted algebra.

1 Many-sorted and Order-sorted Algebras

We recall in this section most of the basic definitions on many-sorted and order-sorted algebras. The reader is referred to [GM87,SNGM87] for more details. We also introduce a notion of equivalent specifications.

1.1 Many-sorted and Order-Sorted Signatures

A Many-Sorted Signature (MSS for short) is a pair (S, F) where S is a set of*sorts* names (which will be denoted $\underline{s}, \underline{s}_1, \ldots$) and F is a set of function symbols together with a typing function τ which associates to each $f \in F$ a string in S^+. When $\tau(f) = \underline{s}_1\underline{s}_2 \ldots \underline{s}_n\underline{s}$ we write $f : \underline{s}_1 \times \ldots \times \underline{s}_n \to \underline{s}$ and say that f has *profile* $\underline{s}_1 \times \ldots \times \underline{s}_n \to \underline{s}$.

An Order-Sorted Signature (OSS for short) is a triple $(S, \geq, F)$ where S is a set of sort symbols, $\geq$ is an ordering on S and F is a set of function symbols together with a typing function τ which associates to each $f \in F$ a finite non empty subset of S^+. All words in $T(F)$ have the same length $n+1$ and $|f| = n$ is the *arity* of f.As in the many-sorted case, we say that f has profile $\underline{s}_1 \times \ldots \times \underline{s}_n \to \underline{s}$ when $\underline{s}_1 \ldots \underline{s}_n\underline{s} \in \tau(f)$.

In both cases (many-sorted and order-sorted) X is a set of *variable* symbols. A sort is assigned to each variable and we write $x : \underline{s}$ $\tau(x) = \underline{s}$. We assume that there are infinitely many variables of each sort.

In both cases, if SIG is a signature, $T(SIG, X)$ (sometimes written $T(F, X)$ when there is no ambiguity) is the set of "well formed" terms constructed on SIG and X in the usual way (cf [GM87] for example). When X is empty we write $T(SIG)$ (or $T(F)$) instead of $T(SIG, \emptyset)$.

In the following, we always assume that, for every $\underline{s} \in S$, there is at least one $t \in T(SIG)$ such that t has sort $\underline{s}$.

A signature is *finite* when both S and F are finite. An OSS is *regular* when each term $t \in T(SIG, X)$ has a least sort $LS(t)$. This property can be syntactically characterized for finite signatures ([GM87] for example): $(S, \geq, F)$ is regular iff, for every $w_0, w_1, w_2 \in S^*$ such that $w_0 \leq w_1$ [2] and $w_0 \leq w_2$ and every $f \in F$ such that $f : w_1 \to \underline{s}_1$ and $f : w_2 \to \underline{s}_2$, there exists $w_3 \in S^*$ and $\underline{s}_3 \in S$ such that $w_0 \leq w_3 \leq w_1, w_2$, $\underline{s}_3 \leq \underline{s}_1, \underline{s}_2$ and $w_3 \to \underline{s}_3 \in \tau(f)$.

If, in addition, each connected component of $(S, \geq)$ has a top element, then the regular signature is called *coherent* ([GM87]). *All signatures considered here are assumed to be coherent and finite* (except when the contrary is explicitly stated).

[2] the ordering on S is extended to S^* by comparing the sorts componentwise

1.2 Order-Sorted Algebras

Let $SIG = (S, \geq, F)$ be an OSS. A (SIG-)Order-Sorted Algebra (OSA for short) $\mathcal{A}$ is (as defined in [SNGM87])

- a family $(\mathcal{A}_{\underline{s}})_{\underline{s} \in S}$ of non empty sets such that, when $\underline{s} \leq \underline{s}'$, then $\mathcal{A}_{\underline{s}} \subseteq \mathcal{A}_{\underline{s}'}$. Let $C_{\mathcal{A}} = \bigcup_{\underline{s} \in S} \mathcal{A}_{\underline{s}}$.
- for each function symbol f, a mapping $f_{\mathcal{A}}$ from $D_f^{\mathcal{A}} \subseteq C_{\mathcal{A}}^{|f|}$ into $C_{\mathcal{A}}$ such that, if f has the profile $w \rightarrow \underline{s}$, then $\mathcal{A}_w \subseteq D_f^{\mathcal{A}}$[3]. and $f(\mathcal{A}_w) \subseteq \underline{s}_{\mathcal{A}}$.

Given a MSS (S, F), an (S, F)-Many-Sorted Algebra is simply an $(S, \geq, F)$ OSA where $\geq$ is the trivial ordering on S ($\underline{s} \geq \underline{s}'$ iff $\underline{s} = \underline{s}'$). In this way, OSAs strictly generalizes MSAs[4]. Therefore, when we speak about substitutions, rewrite rules, ... without any more specific mention, one should understand "order-sorted substitutions", "order-sorted rewrite rules",...

Homomorphisms are defined in the usual way. Then, for any OSS SIG, $T(SIG)$ is an initial OSA.

Let $\mathcal{A}$ be a SIG-OSA. An $\mathcal{A}$-assignment σ is a morphism from $T(SIG, X)$ into $\mathcal{A}$ which associates with each $x : \underline{s}$ an element $t \in \mathcal{A}_{\underline{s}}$.

A substitution σ is a $T(SIG, X)$-assignment such that $Dom(\sigma) = \{x \in X, x\sigma \neq x\}$ (called the *domain* of σ) is finite. The set of SIG-substitutions is denoted by Σ_{SIG} (or simply Σ when there is no ambiguity). If X_0 is a finite subset of X, a X_0-grounding substitution σ is a substitution whose domain includes X_0 and such that $\forall x \in X_0, x\sigma \in T(SIG)$. Often, we will omit the X_0 prefix, assuming that X_0 contains the variables occurring in the terms to which σ is applied. The set of all grounding substitutions w.r.t. some understood X_0 is denoted by $\Sigma_{SIG,g}$ (or simply Σ_g when there is no ambiguity).

1.3 Equations and Rewrite Rules

An *equation* is a pair of terms $s, t \in T(SIG, X)$ where $LS(s)$ and $LS(t)$ are in the same connected component of $(S, \geq)$. A model of a finite set of equations (axioms) E is defined as usual. The class of models of E is referred to as the equational theory E. In [GM87] (for example) a complete set of inference rules for equational deduction is given. This means that every equation which is valid in the equational theory can be derived using these rules. This allows to construct the congruence relation $=_E$ over $T(SIG, X)$ defined by a finite set of equations E. Then we have the following result:

Theorem 1 [GM87] *If SIG is a coherent signature and E a set of equations, then $T(SIG)/=_E$ is initial in the category of models of E.*

[3]If $\underline{s}_1 \ldots \underline{s}_n = w \in S^+$, $\mathcal{A}_w$ is the cartesian product $\mathcal{A}_{\underline{s}_1} \times \ldots \times \mathcal{A}_{\underline{s}_n}$.

[4]Because, here, MSS do not allow "overloaded" declarations.

An Order-Sorted Specification (OSSpec) is a pair (SIG, E) where SIG is an OSS and E a finite set of equations $s = t$ where $s, t \in T(SIG, X)$. A Many-Sorted Specification (MSSpec) is defined in the same way.

A *rewrite rule* is a couple of terms $s, t \in T(SIG, X)$ such that $Var(t) \subseteq Var(s)$. It is written $s \to t$. A Term Rewriting System (TRS) is a finite set of rewrite rules. A TRS $\mathcal{R}$ is *sort decreasing* [KKM88] if, for every rule $s \to t$ in $\mathcal{R}$ and every substitution σ, $LS(s\sigma) \geq LS(t\sigma)$[5]. In such a case, the reduction relation $\to_{\mathcal{R}}$ associated with a TRS $\mathcal{R}$ is defined as in the many-sorted case. $\to^*_{\mathcal{R}}$ is the reflexive transitive closure of $\to_{\mathcal{R}}$. For every relation $\to$, $\leftrightarrow$ is the symmetric closure of $\to$.

A TRS is *noetherian* if there are no infinite chain $t_1 \to_{\mathcal{R}} \dots t_n \to_{\mathcal{R}} \dots$. It is *confluent* (resp. *ground confluent*) if, for all $s, t_1, t_2 \in T(SIG, X)$ (resp. $T(SIG)$), $s \to^*_{\mathcal{R}} t_1$ and $s \to^*_{\mathcal{R}} t_2$ implies the existence of a term u such that $t_1 \to^*_{\mathcal{R}} u$ and $t_2 \to^*_{\mathcal{R}} u$. A TRS $\mathcal{R}$ is *convergent* (resp. *ground convergent*) if it is noetherian and confluent (resp. ground confluent). When a TRS is convergent (resp. ground convergent), for every term $t \in T(SIG, X)$ (resp. $t \in T(SIG)$) there is a unique term $t\downarrow_{\mathcal{R}}$ such that $t \to^*_{\mathcal{R}} t\downarrow_{\mathcal{R}}$ and $t\downarrow_{\mathcal{R}}$ is irreducible by $\mathcal{R}$.

A TRS $\mathcal{R}$ is *canonical* if it is convergent and if for every rule $l \to r$ in $\mathcal{R}$, l and r are irreducible by $\to_{\mathcal{R}-\{l\to r\}}$.

$=_{\mathcal{R}}$ is the congruence on $T(SIG, X)$ generated by the set of axioms obtained by considering the rules in $\mathcal{R}$ as equations. Then $I(\mathcal{R})$ (or $I(E)$) is another notation for the initial algebra $T(SIG)/=_{\mathcal{R}}$. $=_{I(\mathcal{R})}$ is the congruence relation defined on $T(SIG, X)$ by :

$$s =_{I(\mathcal{R})} t \quad \Leftrightarrow \quad \forall \sigma \in \Sigma_g, s\sigma =_{\mathcal{R}} t\sigma$$

1.4 Equivalent Specifications

Let $SIG = (S, \geq, F)$ and $SIG' = (S', \geq', F')$ be two coherent OSS such that $S \subseteq S'$, $\geq \subseteq \geq'$, $F \subseteq F'$ and. for each $f \in F$, $\tau(f) \subseteq \tau'(f)$. Then $T(SIG', X)$ is (canonically) a SIG-algebra. Let ϕ be the unique (injective) SIG-homomorphism from $T(SIG, X)$ into $T(SIG', X \cup X')$ which is the identity on X. Then, the OSSpec (SIG', E') is said to be *equivalent*[6] to (SIG, E) if

$$\forall s, t \in T(SIG, X), \quad (s =_{I(E)} t \quad \Leftrightarrow \quad \phi(s) =_{I(E')} \phi(t))$$

This means that, when we only consider the terms built on SIG, the specifications have the same class of inductive theorems.

Finally, an OSSpec $((S, \geq, F), E)$ is said to be *decomposed* if F can be split into two sets C and D such that:

- $\forall s, t \in T(C), s \neq_E t$
- $\forall s \in T(F) - T(C), \exists t \in T(C), s =_E t$

[5]Note that Rewriting Systems are always sort decreasing in MSA.

[6]This "equivalence" is not symmetric. This is an abbreviation for (SIG, E) is (SIG, E)-equivalent to (SIG', E'), the (SIG, E)-equivalence being indeed symmetric.

2 Transformation of Specifications

In this section we show how to transform a MSSpec, the *source specification* into an equivalent decomposed OSSpec: the *target specification.* However, as the simplification of equational problems can be generalized to finite coherent order-sorted signatures [Com88b], the method given in this section also applies to (finite coherent) OSSpec.

The source specification will be denoted by $(SIG_0, \mathcal{R}_0)$ where $SIG_0 = (S_0, F_0)$ and the target specification by $(SIG_T, \mathcal{R}_T)$ where $SIG_T = (S_T, \geq_T, F_T)$. We assume in the following that $\mathcal{R}_0$ is ground convergent. NF will denote the set of ground terms in $T(F_0)$ that are irreducible by $\mathcal{R}_0$.

2.1 Ground Normal Form Grammars

We don't give here the full algorithm that produces a conditional grammar for NF. Let us only sketch on an example the way it is computed.

Example 1
$F_0 = \{s : int2 \rightarrow int2;\ 0 : \rightarrow int2;\ + : int2 \times int2 \rightarrow int2\}$
$\mathcal{R}_0 = \{\ 0 + x \rightarrow x \quad x + 0 \rightarrow x \quad x + x \rightarrow 0 \quad s(s(0)) \rightarrow 0\ \}$
The first set of derivation rules only states that a term in NF has a root symbol in F_0:

$$NF_{int2} \quad \rightarrow \quad NF_0 \ \mid\ NF_{s(x)} \ \mid\ NF_{x_1+x_2}$$

where NF_t denotes both a non terminal and the language it generates: $NF_t = NF \cap \{t\sigma,\ \sigma \in \Sigma_g\}$.

Now we compute the derivation rules, say, for $NF_{s(x)}$: [7]

$$t \in NF_{s(x)} \quad \text{iff} \quad t = s(u), t \neq s(s(0)) \quad \text{and} \quad u \in NF$$

Solving $s(u) \neq s(s(0))$ in $T(F_0)$, using the algorithm described in [CL88], leads to the four disjoint solutions:

1. $\exists x_1, x_2,\ u = x_1 + x_2$
2. $u = 0$
3. $\exists x_1, x_2,\ u = s(x_1 + x_2)$
4. $\exists x_1,\ u = s(s(x_1))$

This can be expressed by the four rules:

$$NF_{s(x)} \quad \rightarrow \quad s(NF_{x_1+x_2}) \ \mid\ s(NF_0) \ \mid\ s(NF_{s(x_1+x_2)}) \ \mid\ s(NF_{s(s(x))})$$

Using again the same method, we compute the derivation rules for the non terminals we introduced. There would here remain to compute the derivation rules for NF_0, $NF_{x_1+x_2}$, $NF_{s(s(x))}$, $NF_{s(x_1+x_2)}$.

[7]Informally, t is an irreducible ground instance of $s(x)$ iff

1. its root symbol is s
2. it does not match at the root any left hand side of a rule
3. its proper subterms are irreducible

This characterization of $NF_{s(x)}$ can be generalized to any NF_t (see [CR87,Com88a]).

Theorem 2 *This procedure fully described in [Com88a] does always terminate.*

In our example, we get the additional grammar rules:

$$\begin{array}{lcll}
NF_0 & \rightarrow & 0 & \\
NF_{x_1+x_2} & \rightarrow & NF_{s(x_1)} + NF_{s(x_2)} \text{ IF } x_1 \neq x_2 & \mid NF_{x_1+x_2} + NF_{s(x)} \\
 & & \mid NF_{x_1+x_2} + NF_{x_3+x_4} \text{ IF } x_1 \neq x_3 & \mid NF_{s(x)} + NF_{x_1+x_2} \\
 & & \mid NF_{x_1+x_2} + NF_{x_3+x_4} \text{ IF } x_2 \neq x_4 & \\
NF_{s(s(x))} & \rightarrow & s(NF_{s(s(x))}) & \mid s(NF_{s(x_1+x_2)}) \\
NF_{s(x_1+x_2)} & \rightarrow & s(NF_{x_1+x_2}) &
\end{array}$$

Then the grammar is "cleaned up" using an algorithm described in [Com88b,Com88a]. This algorithm is similar to the usual cleaning algorithm for context free word grammars; the non terminals from which there is no derivation chain reaching a terminal tree (called *useless* non terminals) are removed.

Theorem 3 [Com88b,Com88a] *There is an algorithm producing a conditional grammar of NF which does not contain any useless non-terminal.*

The grammar produced in this way will be called the *reduced grammar of NF*. In our example, we get the following reduced grammar:

$$\begin{array}{rcll}
NF_{int2} & \rightarrow & NF_0 & \mid NF_{s(x)} \\
NF_{s(x)} & \rightarrow & s(NF_0) & \\
NF_0 & \rightarrow & 0 &
\end{array}$$

And, indeed, there are only two terms in NF : 0 and $s(0)$.

Of course, this step (cleaning the grammar) is equivalent to an inductive reducibility test since we proved simultaneously that $x_1 + x_2$, $s(x_1 + x_2)$ and $s(s(x))$ are inductively reducible (the corresponding set of irreducible ground instances are empty). However, this computation has to be done only once, whatever inductive completion is performed afterwards.

We give another simple example which illustrates the transformation. This is a specification of the integers.

Example 2

$$\begin{array}{l}
F = \{ \; s, p : int \rightarrow int; \; 0 : \rightarrow int \quad + : int \times int \rightarrow int \; \} \\
\mathcal{R} = \{ \begin{array}{rcl}
s(p(x)) & \rightarrow & x \\
p(s(x)) & \rightarrow & x \\
0 + x & \rightarrow & x \\
s(x) + y & \rightarrow & s(x + y) \\
p(x) + y & \rightarrow & p(x + y) \; \}
\end{array}
\end{array}$$

We get the following reduced grammar for NF:

$$\begin{array}{rclll}
NF & \rightarrow & NF_0 & \mid NF_{s(x)} & \mid NF_{p(x)} \\
NF_0 & \rightarrow & 0 & & \\
NF_{s(x)} & \rightarrow & s(NF_0) & \mid s(NF_{s(x)}) & \\
NF_{p(x)} & \rightarrow & p(NF_0) & \mid p(NF_{p(x)}) &
\end{array}$$

In both examples, the reduced grammar of NF is a regular tree grammar. *We will assume this property in the following.* Note that such a property is ensured by the left linearity of the original TRS (see e.g. [GB85]). However this condition is not necessary as shown by example 1.

More precisely, we call NF-grammar any pair $\mathcal{G} = (NT, P)$ satisfying

- $NT = \{NF_{\underline{s}} | \underline{s} \in S\} \cup \{NF_t | t \in T_0\}$ where T_0 is a finite subset of linear terms in $T(F_0, X)$[8].
- P is a set of derivation rules $N \to f(N_1, \ldots, N_k)$ or $N \to N'$ such that:
  1. $N, N', N_1, \ldots, N_k \in NT$
  2. $f \in F_0$
  3. $\forall N \in NT, N = \bigcup_{N \to N' \in P} N'$
  4. $\forall NF_t, NF_{t'} \in NT$, t is not a variable and t and t' are not equal up to the renaming of their variables.
  5. For each $N \to f(N_1, \ldots, N_k) \in P$, $N = NF_t$ for some t such that $root(t) = f$.
  6. For each $N \to N' \in P$, $N = NF_{\underline{s}}$ for some $\underline{s} \in S_0$
  7. For each $N \in NT$, $N \neq \emptyset$

The reduced grammar of NF is an NF-grammar.

2.2 Constructing $(S_T, \geq_T)$

Let $\mathcal{G} = (NT, P)$ be an NF-grammar. S_T is the set of non terminals NT. For sake of clarity, we rename the terms in NT: in example 2, $NF_{s(x)}$ is usually denoted by *pos* (for strictly positive integers) and $NF_{p(x)}$ is usually denoted by *neg*.

The ordering $\geq_T$ on S_T is defined by:

- If $t, t' \in T(F_0, X)$, $NF_t \geq_T NF_{t'}$ iff $\exists \sigma \in \Sigma, t' = t\sigma$.
- $NF_{\underline{s}} \geq_T NF_t$ when $LS(t) \leq_0 \underline{s}$

F_0 is split into the sets C_0 and D_0 in the following way:

- C_0 is the set of function symbols f such that there is a rule $NF_{\underline{s}} \to NF_{f(x_1,\ldots,x_n)}$ in P
- $D_0 = F - C_0$

Equivalently, D_0 is the set of symbols $f \in F$ such that $f(x_1, \ldots, x_n)$ is inductively reducible. (This is so because of the properties of $\mathcal{G}$).

Now, let C_T be the set C_0 where every symbol has been primed. For each production rule $N \to f(N_1, \ldots, N_k)$ we associate with $f' \in C_T$ the profile $f' : N_1 \times \ldots \times N_k \to N$. We get now an OSS $(S_T, \geq_T, C_T)$[9] Let us show how it works on our two examples:

[8]When NF_t is computed by the algorithm, then t is always a linear term. Thus this is not an additional assumption. (See [Com88b,Com88a]).

[9]Note that condition 7 in the definition of an NF-grammar ensures that, for every $\underline{s} \in S_T$, $T(C_T)_{\underline{s}}$, the set of ground terms of sort $\underline{s}$ is not empty (as we required in the definition of an OSS)

Example 1
We associate with each non terminal a sort in the target specification. In this example, *int2* is associated with NF_{int2}, *pos* with $NF_{s(x)}$ and *zero* with NF_0. Then, each rule of the reduced grammar corresponds either to a subsort declaration or a profile declaration. The rules

$$NF \rightarrow NF_0 \mid NF_{s(x)}$$

give the inclusions $int2 > pos$ and $int2 > zero$.
The rule $NF_{s(x)} \rightarrow s(NF_0)$ gives $s' : zero \rightarrow pos$ and the rule $NF_0 \rightarrow 0$ gives $0' :\rightarrow zero$.

Example 2
This leads to the sort structure $S_T = \{int, pos, neg, zero\}$ with the relations $int > neg$, $int > pos$ and $int > zero$ corresponding to the rules

$$NF \rightarrow NF_0 \mid NF_{s(x)} \mid NF_{p(x)}$$

The other rules correspond to the profile declarations:

$$0' :\rightarrow zero \qquad \begin{array}{ll} s' : & zero \rightarrow pos \\ & pos \rightarrow pos \end{array} \quad \begin{array}{ll} p' : & zero \rightarrow neg \\ & neg \rightarrow neg \end{array}$$

Proposition 1 *Assume that $\mathcal{R}_0$ is ground convergent, then NF is an $(S_T, \geq_T, C_T)$-algebra*

In general $(S_T, \geq_T, C_T)$ is not coherent. In order to guarantee its coherence, we have to construct $(S_T, \geq_T, C_T)$ using a suitable NF-grammar:

Proposition 2 *Assume that $\mathcal{R}_0$ is ground convergent and that there exists an NF-grammar. Then there exists an NF-grammar $\mathcal{G}$ such that $(S_T, \geq_T, C_T)$ is a coherent OSS.*

Sketch of the proof: the NF-grammar is constructed starting from any NF-grammar $\mathcal{G}_0$ and adding some sorts and some profiles in the following way : for every pair of rules

$$\begin{array}{lcl} NF_{f(t_1,\ldots,t_n)} & \rightarrow & f(NF_{u_1}, \ldots, NF_{u_n}) \\ NF_{f(t'_1,\ldots,t'_n)} & \rightarrow & f(NF_{u'_1}, \ldots, NF_{u'_n}) \end{array}$$

such that

- for all i, u_i and u'_i are unifiable with a most general common instance $u_i \wedge u'_i$
- $NF_{u_i \wedge u'_i} \in NT_0$
- $\exists i$ s.t. u_i is not an instance of u'_i and $\exists j$ s.t. u'_j is not an instance of u_j

add the grammar rule

$$NF_{f(t_1 \wedge t'_1,\ldots,t_n \wedge t'_n)} \rightarrow f(NF_{u_1 \wedge u'_1}, \ldots, NF_{u_n \wedge u'_n})$$

and the non terminal $f(t_1 \wedge t'_1, \ldots, t_n \wedge t'_n)$ (if not already in NT_0).
It is not difficult to see that the resulting set of non terminals and production rules constitute an NF-grammar. (Although some non-terminals may not be reachable). With such an NF-grammar, $(S_T, \geq_T, C_T)$ is coherent.

2.3 Computing the target specification

We take $D_T = F$ and $F_T = D_T \cup C_T$ together with the profile declarations $f : NF_{\underline{s}_1} \times \ldots \times NF_{\underline{s}_n} \to NF_{\underline{s}}$ if $f \in F$ has the profile $\underline{s}_1 \times \ldots \times \underline{s}_n \to \underline{s}$.

Then, each term in $T(F_0, X_0)$ can be viewed as a term in $T(F_T, X_T)$. In other words:

Lemma 1 *$T(F_T, X_T)$ is a (free) (S_0, F_0)-algebra.*

Indeed, let $[\![\cdot]\!]$ denote the function defined on $T(F_0, X_0)$ by:

- $[\![x : \underline{s}]\!] = x : NF_{\underline{s}}$
- $[\![f(t_1, \ldots, t_n)]\!] = f([\![t_1]\!], \ldots, [\![t_n]\!])$

$[\![\cdot]\!]$ is an injective (S_0, F_0)-morphism.

Let $\mathcal{R}_1$ be the set of rules

$$f(x_1 : \underline{s}_1, \ldots, x_n : \underline{s}_n) \to f'(x_1 : \underline{s}_1, \ldots, x_n : \underline{s}_n)$$

for every $f \in C_0$ and every profile $f' : \underline{s}_1 \times \ldots \times \underline{s}_n \to \underline{s}$. Such a construction is "well formed" since, if $f : \underline{s}'_1 \times \ldots \times \underline{s}'_n \to \underline{s}'$ in SIG_0, then, for every index i, $\underline{s}_i \geq_T NF_{\underline{s}'_i}$.

Lemma 2 *$\mathcal{R}_1$ is canonical and sort decreasing.*

This is indeed a consequence of proposition 2.

Let $\mathcal{R}_2$ be the set of rewrite rules $[\![t]\!] \downarrow_{\mathcal{R}_1} \to [\![u]\!] \downarrow_{\mathcal{R}_1}$ for every rule $t \to u$ in $\mathcal{R}_0$.

A *decreasing renaming* of a term t is a substitution θ which associates to each variable a variable with a lower sort in such a way that there is at least one variable x in t such that $sort(x\theta) < sort(x)$.

$\mathcal{R}_T = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_2^*$ where $\mathcal{R}_2^*$ is the set of rules $l\theta \downarrow_{\mathcal{R}_1} \to r\theta \downarrow_{\mathcal{R}_1}$ for each rule $l \to r \in \mathcal{R}_2$ and each decreasing renaming of l. (Such a set of rules is finite as S is finite). Of course, rules in $\mathcal{R}_T$ which are instances of some other rule in $\mathcal{R}_T$ can be removed.

Example 1 [10]

$$\begin{array}{rcl}
\mathcal{R}_1 = \{ & s(x : zero) \to s'(x : zero) & \\
 & 0 \to 0' & \} \\
\mathcal{R}_2 = \{ & s(s'(0')) \to 0' & \\
 & 0' + x \to x & \\
 & x + 0' \to x & \\
 & x + x \to 0' & \}
\end{array}$$

and $\mathcal{R}_T = \mathcal{R}_1 \cup \mathcal{R}_2$ (every rule in $\mathcal{R}_2^*$ is an instance of a rule in $\mathcal{R}_2$).

[10] When the sort of a variable is not mentioned, it must be understood that it has the greatest sort of its connected component.

Example 2

$$
\begin{array}{llll}
\mathcal{R}_1 = \{ & s(x:zero) \to s'(x:zero) & s(x:pos) \to s'(x:pos) & \\
& p(x:zero) \to p'(x:zero) & p(x:neg) \to p'(x:neg) & \\
& 0 \to 0' & & \} \\
\mathcal{R}_2 = \{ & s(p(x)) \to x & p(s(x)) \to x & \\
& 0' + x \to x & s(x) + y \to s(x+y) & \\
& p(x) + y \to p(x+y) & & \} \\
\mathcal{R}_2^* = \{ & s(p'(x:zero)) \to x:zero & s(p'(x:neg)) \to x:neg & \\
& p(s'(x:zero)) \to x:zero & p(s'(x:pos)) \to x:pos & \\
& s'(x:zero) + y \to s(x+y) & s'(x:pos) + y \to s(x+y) & \\
& p'(x:zero) + y \to p(x+y) & p'(x:neg) + y \to p(x+y) & \}
\end{array}
$$

2.4 Properties of the target specification

There are basically two mappings linking the source and the target specification. $[\![\cdot]\!]$ has already been mentioned. Now, for $t \in T(SIG_T, X)$ let $\tilde{t}$ be the term in $T(SIG_0, X_0)$ obtained by replacing each variable x by a variable x' whose sort is the greatest sort in the connected component of $sort(x)$ and replacing each primed function symbol by the unprimed one. (Then $[\![\tilde{t}]\!] = t$).

Our construction using NF-grammars has the following main property:

Lemma 3 *$u \to_{\mathcal{R}_T} v$ iff $u \to_{\mathcal{R}_1} v$ or $\tilde{u} \to_{\mathcal{R}_0} \tilde{v}$.*

Corollary 1 *For every $t \in T(SIG_T)$, $\tilde{t}$ is irreducible by $\mathcal{R}_0$ iff t is irreducible by $\mathcal{R}_2 \cup \mathcal{R}_2^*$.*

Corollary 2 *For every $t, u \in T(SIG_T)$, $\tilde{t}\downarrow_{\mathcal{R}_0} = \tilde{s}\downarrow_{\mathcal{R}_0}$ iff $t\downarrow_{\mathcal{R}_T} = s\downarrow_{\mathcal{R}_T}$*

By construction, the target OSSpec is coherent[11]. The rewrite system $\mathcal{R}_T$ has also the desired properties:

Lemma 4 *If $\mathcal{R}_0$ is ground convergent, then so is $\mathcal{R}_T$.*

Sketch of the proof
When $t \to_{\mathcal{R}_T} u$, either $\tilde{t} \to_{\mathcal{R}_0} \tilde{u}$ or $\tilde{t} = \tilde{u}$. In the latter case $t \to_{\mathcal{R}_1} u$. This proves that $\mathcal{R}_T$ does terminate. The ground confluence proof is more involved; assuming that $s, t_1, t_2 \in T(SIG_T)$ and $s \to_{\mathcal{R}_T} t_1$ and $s \to_{\mathcal{R}_T} t_2$, there are three cases to investigate:

1. $\tilde{s} = \tilde{t_1} = \tilde{t_2}$. Then we use lemma 2
2. $\tilde{s} = \tilde{t_1} \neq \tilde{t_2}$. Then we use the construction of $\mathcal{R}_T$
3. $\tilde{t_1} \neq \tilde{s} \neq \tilde{t_2}$. Then we use the ground confluence of $\mathcal{R}_0$.

Lemma 5 *$\mathcal{R}_T$ is sort-decreasing.*

[11]This is easy to deduce from proposition 2. Note that this proves the existence of the initial algebra, as recalled in section 1.

This can be easily verified. As shown in [KKM88], when a term rewriting system is convergent and sort decreasing, for every equational order-sorted deduction of $s = t$, there is a rewrite proof of $s = t$. This result easily extends to ground convergent TRS:

Lemma 6 *If $\mathcal{R}$ is ground convergent and sort decreasing, then $s =_{I(\mathcal{R})} t$ iff, for every grounding substitution σ $s\sigma \downarrow_{\mathcal{R}} = t\sigma \downarrow_{\mathcal{R}}$.*

Now what we expect for inductive proofs is the equivalence between the two specifications:

Theorem 4 *If $\mathcal{R}_0$ is ground convergent, then $(SIG_T, \mathcal{R}_T)$ is equivalent to $(SIG_0, \mathcal{R}_0)$.*

In other words

$$s =_{I(\mathcal{R}_0)} t \quad \Leftrightarrow \quad [\![s]\!] =_{I(\mathcal{R}_T)} [\![t]\!]$$

Sketch of the proof

When $\sigma \in \Sigma_{SIG_0}$, $[\![\sigma]\!]$ is the substitution whose domain is $Dom(\sigma)$ and such that, for every $x \in Dom(\sigma)$, $x[\![\sigma]\!] = [\![x\sigma]\!]$. In the same way, when $\sigma \in \Sigma_{SIG_T}$, $\tilde{\sigma}$ is defined by $Dom(\tilde{\sigma}) = \{\tilde{x}, x \in Dom(\sigma)\}$ and, for every $x \in Dom(\sigma)$, $\tilde{x}\tilde{\sigma} = \widetilde{x\sigma}$. Now the theorem follows from the equivalences:

- $[\![s]\!] =_{I(\mathcal{R}_T)} [\![t]\!] \quad \Leftrightarrow \quad \forall \sigma \in \Sigma_{SIG_{T,g}},\ s\tilde{\sigma} \downarrow_{\mathcal{R}_0} = t\tilde{\sigma} \downarrow_{\mathcal{R}_0}$
- $s =_{I(\mathcal{R}_0)} t \quad \Leftrightarrow \quad \forall \sigma \in \Sigma_{SIG_{0,g}},\ [\![s]\!][\![\sigma]\!] \downarrow_{\mathcal{R}_T} = [\![t]\!][\![\sigma]\!] \downarrow_{\mathcal{R}_T}$

Let us show the second equivalence. $s =_{I(\mathcal{R}_0)} t$ iff for every grounding substitution σ, $s\sigma \downarrow_{\mathcal{R}_0} = t\sigma \downarrow_{\mathcal{R}_0}$. On the other hand, $s\sigma \downarrow_{\mathcal{R}_0} = \widetilde{[\![s\sigma]\!]} \downarrow_{\mathcal{R}_0}$ and, by corollary 2, $\widetilde{[\![s\sigma]\!]} \downarrow_{\mathcal{R}_0} = \widetilde{[\![t\sigma]\!]} \downarrow_{\mathcal{R}_0}$ iff $[\![s\sigma]\!] \downarrow_{\mathcal{R}_T} = [\![t\sigma]\!] \downarrow_{\mathcal{R}_T}$. Now, the equivalence follows from the identity $[\![s\sigma]\!] = [\![s]\!][\![\sigma]\!]$.

For the first equivalence, by lemmas 4 and 6, $[\![s]\!] =_{I(\mathcal{R}_T)} [\![t]\!]$ iff for every grounding substitution σ, $[\![s]\!]\sigma \downarrow_{\mathcal{R}_T} = [\![t]\!]\sigma \downarrow_{\mathcal{R}_T}$. Now, following the identity $[\![s]\!]\sigma \downarrow_{\mathcal{R}_T} = [\![s\tilde{\sigma}]\!] \downarrow_{\mathcal{R}_T}$, we have the equivalences

$$\begin{array}{rcl} [\![t]\!]\sigma \downarrow_{\mathcal{R}_T} = [\![s]\!]\sigma \downarrow_{\mathcal{R}_T} & \Leftrightarrow & [\![s\tilde{\sigma}]\!] \downarrow_{\mathcal{R}_T} = [\![t\tilde{\sigma}]\!] \downarrow_{\mathcal{R}_T} \\ & \Leftrightarrow & \widetilde{[\![s\tilde{\sigma}]\!]} \downarrow_{\mathcal{R}_0} = \widetilde{[\![t\tilde{\sigma}]\!]} \downarrow_{\mathcal{R}_0} \\ & \Leftrightarrow & s\tilde{\sigma} \downarrow_{\mathcal{R}_0} t\tilde{\sigma} \downarrow_{\mathcal{R}_0} \end{array}$$

□

It is thus possible to perform inductive proofs in the target algebra instead of the source algebra. As announced, we have also the following property which states that the target algebra is "simpler" than the source one:

Proposition 3 *$(SIG_T, \mathcal{R}_T)$ is a decomposed OSSpec.*

This indeed is a consequence of lemma 3.

3 Inductive Proofs in Order-Sorted Algebras

Now, it remains to show how to perform inductive proofs in decomposed order-sorted algebras. The aim of this paper is not to give results in this field. Therefore, we only show how to perform inductive proofs in our target algebra and sketch a general method.

The only difficulty with OSA is that equational reasoning may lead to ill formed terms (see for example [SNGM87]). Such a problem does not occur when dealing with sort decreasing term rewriting systems. And, by lemma 5, our system is sort-decreasing.

However, if we use the order-sorted completion (as in [KKM88]), an equational consequence $u = v$ where neither $LS(u) \leq LS(v)$ nor $LS(v) \leq LS(u)$ may be derived. In such a case, it is not possible to orient the equation and keep the sort-decreasing property.

Let us assume that the source algebra is a MSSpec. In this case, the target specification has some additional properties which ensure that such a sort problem cannot occur:

Lemma 7 *Assume that the source specification is a MSSpec. Let $s = t$ be an equation in the source specification. Then every equational consequence $u = v$ of $\mathcal{R}_T \cup \{[\![s]\!] = [\![t]\!]\}$ derived by a completion procedure satisfies either $LS(u\sigma) \geq LS(v\sigma)$ or $LS(v\sigma) \geq LS(u\sigma)$ for every substitution σ or $u, v \in T(C_T, X)$.*

This indeed can easily be proved using the fact that a term whose root symbol is in D_T has necessarily a sort which is maximal in its connected component[12].

Now three situations can occur when an equation $u = v$ is derived by the (inductive) completion procedure :

1. $u, v \in T(C_T, X)$ and it is possible to derive a contradiction
2. for every substitution σ, $LS(u\sigma) > LS(v\sigma)$ (or $LS(v\sigma) > LS(u\sigma)$) in which case the equation can be oriented, provided that constructor terms are smaller than non constructor ones for the reduction ordering.
3. for every substitution σ, $LS(u\sigma) = LS(v\sigma)$

Therefore, it is possible to use completion procedures (as in [KKM88]) in this case, without modifying the sort structure.

When the source specification is an OSSpec there are some more difficulties since an equation between two non constructor terms with uncomparable sorts can be derived by a completion procedure.

However, in order to solve complement problems in OSA, it is necessary to transform the sort structure ([Com88b]). In the result of this transformation (which is mainly a tree automaton determinization), two distinct sorts have disjoint carriers in $T(F)$. (But functions symbols remain "overloaded"). If we assume this additional property of the source specification, then lemma 7 holds for order sorted specifications. Therefore, no sort problem can occur during a completion procedure.

Examples 1 and 2

In examples 1 and 2 (previously defined) it is not possible to use directly the results of [HH82,Lan81] since there are some relations between constructors. However, using the target specification (see above), it is possible to use these methods.

[12]Our target signature may be compared with the *compatible signatures* of [SNGM87].

For example, in example 1, the commutativity $x + y = y + x$ is an inductive theorem since there is no (proper) critical overlap between $x + y$ and a rule in $\mathcal{R}_T$. This is sufficient for ensuring $x + y =_{I(\mathcal{R}_T)} y + x$ as shown in [Bac88].

Let us show how it is proved that $s(x) + y = y + 0$ is not an inductive theorem:

$$\cfrac{\cfrac{\cfrac{s(x) + y = y + 0}{s(x) = 0'}\ \text{overlap with the rule } x + 0' \to x}{s'(x : zero) = 0'}\ \text{overlap with the rule } s(x : zero) \to s'(x : zero)}{\text{DISPROOF}}\ \text{since } s'(x : zero), 0' \in T(C_T, X)$$

4 Concluding remarks

For any term rewriting system it is possible to compute a conditional grammar for NF without useless non-terminals. However, the method presented in this paper requires some more hypothesis. First, it requires the term rewriting system to be ground convergent (may be this hypothesis could be weakened to ground confluence). Secondly, it requires the clean grammar of NF to be a regular one. This means that the language of reducible ground terms is regular. There is no hope to weaken this hypothesis since the set of well formed ground terms in a finite OSS is a regular tree language: profile declarations provide a (bottom-up) finite tree automaton for recognizing it. Therefore, our transformation into a decomposed OSSpec seems to be optimal in some sense: whenever such a transformation exists, then it is computed[13].

A drawback of the method is that there does not exist any simple check for the regularity of the language of reducible ground terms. It is known that left linearity is a sufficient condition (see e.g. [GB85]). However, many examples can be built showing that this condition is not necessary[14]. Of course, it is possible to simply compute the cleaned up grammar and see if it is regular. But, as shown in this paper, this is not an easy computation. An open question is to broaden the left linearity condition in order to have some more general (syntactic) sufficient condition for regularity.

Let us also note that we could not use any regular tree grammar for our transformation since we actually use additional properties of the grammars produced by our algorithm for proving a stronger property of the target specification: the *equivalence* with the source one. Indeed, the isomorphism $T(F)/=_E \sim T(F')/=_{E'}$ does not provide in itself a way for deducing inductive theorems in $T(F, X)$ from inductive theorems in $T(F', X)$.

Anyway, at least in the left linear case (and others, see above), our method proves that it is possible to use Huet and Hullot's algorithm and avoid inductive reducibility checks. This is very useful since the test set of Plaisted's inductive reducibility test is always huge. That is the reason why we think our approach is well suited to the implementation of inductive proofs in equational theories without constructors.

[13]This optimality result will be detailed in a forthcoming paper.

[14]For example $F = \{a, f, h\}$ and the left hand sides are $\{h(f(x,x)), f(f(x_1,x_2),x_3), f(h(x_1),x_2)\}$. The language of reducible ground terms is regular.

Also, it must be noted that inductive proofs in order-sorted decomposed specifications is not harder than in the unsorted case. Indeed, this is the meaning of lemma 7. Therefore, our method is a real improvement over classical ones.

References

[Bac88] L. Bachmair. Proof by consistency in equational theories. In *Proc. 3rd IEEE Symp. Logic in Computer Science, Edinburgh*, July 1988.

[CL88] H. Comon and P. Lescanne. *Equational Problems and Disunification*. Research Report Lifia 82 Imag 727, Univ. Grenoble, May 1988. To appear in J. Symbolic Computation.

[Com88a] H. Comon. An effective method for handling initial algebras. In *Proc. 1st Workshop on Algebraic and Logic Programming, Gaussig*, 1988.

[Com88b] H. Comon. *Unification et Disunification: Théorie et Applications*. Thèse de Doctorat, I.N.P. de Grenoble, France, 1988.

[CR87] H. Comon and J.-L. Remy. *How to Characterize the Language of Ground Normal Forms*. Research Report 676, INRIA, June 1987.

[Fri86] L. Fribourg. A strong restriction of the inductive completion procedure. In *Proc. 13th ICALP, Rennes, LNCS 226*, pages 105–115, Springer-Verlag, 1986.

[GB85] J. H. Gallier and R. V. Book. Reductions in tree replacement systems. *Theoretical Computer Science*, 37:123–150, 1985.

[GM87] J. Goguen and J. Meseguer. *Order-Sorted Algebra I: Partial and Overloaded Operators, Errors and Inheritance*. Draft, Computer Science Lab., SRI International, 1987.

[Gog80] J. A. Goguen. How to prove inductive hypothesis without induction. In *Proc. 5th Conf. on Automated Deduction, LNCS 87*, 1980.

[HH82] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2), 1982.

[JK86] J.-P. Jouannaud and E. Kounalis. Automatic proofs by induction in equational theories without constructors. In *Proc. 1st IEEE Symp. Logic in Computer Science, Cambridge, Mass.*, June 1986.

[KKM88] C. Kirchner, H. Kirchner, and J. Meseguer. Operational semantics of obj-3. In *Proc. 15th Int. Conf on Automata, Languages and Programming, LNCS 317*, Springer-Verlag, July 1988.

[KM86] D. Kapur and R. D. Musser. Inductive reasoning with incomplete specifications. In *Proc. 1st IEEE Symp. Logic in Computer Science, Cambridge, Mass.*, June 1986.

[KM87] D. Kapur and D. Musser. Proof by consistency. *Artificial Intelligence*, 31(2), February 1987.

[KNZ85] D. Kapur, P. Narendran, and H. Zhang. *On Sufficient Completeness and Related Properties of Term Rewriting Systems*. Research Report, General Electric Company, October 1985. Preprint.

[KNZ86] D. Kapur, P. Narendran, and H. Zhang. Proof by induction using test sets. In *Proc. 8th Conf. on Automated Deduction, Oxford, LNCS 230*, Springer-Verlag, July 1986.

[Kuc87] W. Kuchlin. *Inductive Completion by Ground Proofs Transformation.* Research Report, University of Delaware, February 1987.

[Lan81] D. Lankford. *A simple explanation of inductionless induction.* Technical Report MTP-14, Mathematics Department, Louisiana Tech. Univ., 1981.

[Mus80] D. Musser. Proving inductive properties of abstract data types. In *Proc. 7th ACM Symp. Principles of Programming Languages, Las Vegas*, 1980.

[Pla85] D. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.

[SNGM87] G. Smolka, W. Nutt, J. Goguen, and J. Meseguer. *Order-Sorted Equational Computation.* SEKI Report SR-87-14, Univ. Kaiserslautern, December 1987.

[Tha85] S. R. Thatte. On the correspondance between two classes of reductions systems. *Information Processing Letters*, 20:83–85, February 1985.