Deduction with symbolic constraints

Claude Kirchner, Hélène Kirchner, Michaël Rusinowitch

INRIA

UNITÉ DE RECHERCHE
INRIA-LORRAINE

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1358

Programme 1
Programmation, Calcul symbolique et
Intelligence artificielle

DEDUCTION WITH SYMBOLIC CONSTAINTS

Claude KIRCHNER, Hélène KIRCHNER
Michaël RUSINOWITCH

Décembre 1990

RR-1358

Deduction with Symbolic Constraints
Déduction avec Contraintes Symboliques*

Claude Kirchner Hélène Kirchner
Michaël Rusinowitch

INRIA Lorraine: BP 101, 54600 Villers-lès-Nancy, France.
CRIN: BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France.
E-mail: kirchner@loria.crin.fr, rusi@loria.crin.fr

ABSTRACT: A framework for first-order constrained deduction is proposed in this paper. The syntax and semantics of symbolic constraints and constrained formulae are defined. Constrained deduction rules are given for equational logic and for first-order logic with equality. They are applied to clausal theorem proving, unfailing completion and completion modulo a set of axioms.

RÉSUMÉ : Nous proposons un cadre pour la déduction contrainte du premier ordre. La syntaxe et la sémantique des contraintes symboliques et des formules contraintes sont définies. Puis nous donnons des règles de déduction contrainte pour la logique équationnelle et la logique du premier ordre avec égalité. Elles sont ensuite appliquées à la preuve de théorèmes clausals, à la complétion sans échec et à la complétion modulo un ensemble d'axiomes.

PROGRAMME 1

Paru dans la *Revue d'Intelligence Artificielle*, volume 4, numéro 3, pages 9–52, numéro spécial sur la déduction automatique édité par Ricardo Caffera.

*Travail partiellement réalisé dans l'opération ELIOS du GRECO de Programmation et Outils pour l'Intelligence Artificielle

1 Introduction

The process of making visible every symbolic computation step involved in a deduction process is what we call constrained deduction. Constrained formulae are first-order formulae completed by constraints; their semantics is to schematize all the instances of the formula satisfying the constraint part. This technique has many advantages which already have been recognized in the context of logic programming. In the framework of deduction processes too, it is most useful to homogenize the deduction rules by letting apparent the unification, typing and orientation problems. First, this allows studying strategies at low level, refining them and hopefully deriving efficient ones, for instance by delaying the solving of difficult constraints (as CLP(R) currently does with non-linear constraints [HJM*87, JL86]). Second, constraints can account for many aspects of structure-sharing: several instances of a term may be replaced by one constraint. Last, and this is perhaps the main point, constrained deduction is more expressive than classical deduction in the following sense: it makes possible to represent infinitely many objects by a single one. For instance, the constrained equality $[f(x) = f(y), (x \neq y)]$ schematizes an infinite number of classical equalities which cannot be equivalently replaced by a finite number of equalities. This aspect has also been appreciated in [CZ90].

We shall see that this approach has a lot of useful applications in automated theorem proving. It also gives precise tools for solving problems in artificial intelligence, as for example robot motion, where solving constraint problems in computational geometry is a main problem. We focuss here on unification and orientation problems. The proposed deduction rules are very general and we feel that most approaches followed in completion with constraints [Pet90] or in PROLOG with constraints fit into this framework. However we must warn the reader that the inference rules presented here should not be loosely implemented. In order to get well-behaved strategies a careful scheduling should be thought about too. But the nice aspect of this approach is that any kind of operation may be interleaved: for instance one step of a unification algorithm and one step of rewriting. This gives much more choice for controlling the deduction.

The first part of the paper, from Section 2 to 6, states the general framework for constrained deduction. In Section 2, we make precise the symbolic constraints we are interested in, using a general framework proposed by G. Smolka. In Section 3, constrained formulae and their semantics are defined. In Section 4, constrained deduction rules for equational logic are given, while Section 5 deals with constrained deduction rules for first-order logic with equality. For the sake of completeness and efficiency, restructuring rules, given in Section 6, are needed. They make a bridge between the constraints and the first-order part.

The second part of the paper specializes the previous framework to effective procedures. In Section 7, we apply the deduction process to clausal theorem proving and in Section 8, to unfailing completion, by setting more control on the strategy of deduction rule application. With another refinement, we get in Section 9, a completion procedure modulo a set of axioms. These two last

sections show the expressive power of the tools developed here. We conclude with a comparison with other works and current perspectives of this work.

We assume the reader to be familiar with equational logic [Tay79], first-order logic [Gal86, Rob65], class rewriting systems, also called equational term rewriting systems [HO80, Bac87, JK86], and equational problems [Kir85, Com88, CL89, Bur87, KL87]. Our notations are mainly consistent with [DJ90] for the rewriting concepts and [JK90] for the unification part. We refer the reader to these two last papers for a full exposition of the preliminary concepts needed to understand this work.

2 Constraint languages for proofs

In [Smo89], G. Smolka proposes a very general notion of constraint language, together with a method to combine PROLOG-like logic programming with a constraint language. In this section, we first give the general definition of a constraint language and then instantiate it in order to define the particular constraints we are interested in.

2.1 A general constraint language

Definition 1 *[Smo89] A* constraint language *is given by*

- *a countably infinite set V of variables*
- *a set C of constraints c*
- *a function $\mathcal{V}$ from C to V such that $\mathcal{V}(c)$ is the (finite) set of variables constrained by c*
- *a non-empty set of interpretations I. Each interpretation I is given by a non-empty set D_I called the domain (or a family of non-empty sets if sorts are considered), and a solution mapping Sol_I that associates to each c in C the set of its* solutions *in I, denoted by $Sol_I(c)$, that is a subset of the set of assignments $V \mapsto D_I$.*

A constraint c of C is *satisfiable* if there exists at least one interpretation in which c has a solution. It is *valid* in an interpretation I if every assignment is a solution of c in I. A constraint language is *decidable* if the satisfiability of its constraints is decidable.

A symbolic constraint language is built from given sets F of function symbols and P of predicate symbols. The algebra of terms built from F and V is denoted by $\mathcal{T}(F, V)$. We use $\mathcal{T}(F)$ for the algebra of ground terms (i.e. terms with no variables), and $\mathcal{A}(P)$ for the set of ground atoms. The set of variables of a term t and a literal p are respectively denoted $\mathcal{V}(t)$ and $\mathcal{V}(p)$.

Terms will be denoted by letters a, b, s, t, u, v, w, or l, r, g, d. The notation $s[t]$ means that t is a strict subterm of s. When the position m of this subterm needs to be specified, we write $s[t]_m$.

Any interpretation I of variables is extended to a new interpretation, also called I, by leaving the domain of I unchanged and by choosing an interpretation $f_I : D_I^n \rightarrow D_I$ for each f in F, and an interpretation $p_I \subseteq D_I^n$ for each p in P.

Let us now formalize the specific constraint language we are going to use for constrained deduction.

Definition 2 *The* atomic constraint language $ACL[F, P]$ *is defined by:*

- *a set V of variables.*
- *the set C of atomic constraints built on first-order terms $T(F, V)$ and predicates P.*
- *the function $\mathcal{V}$, that associates to a constraint c the set of its constrained variables.*
- *a non-empty set of interpretations I. Each interpretation I is given by a domain D_I and a solution mapping that associates*
  - *to each variable x in C, the set of assignments $Sol_I(x) = \{\alpha : V \mapsto D_I\}$*
  - *to each term $t = f(t_1, ..., t_n)$ in C, the set of assignments*

$$Sol_I(t) = \{\alpha : V \mapsto D_I \mid \underline{\alpha}(t) \in D_I\}$$

where $\underline{\alpha}$ is defined by:
$\underline{\alpha}(f(t_1, ..., t_n)) = f_I(\underline{\alpha}(t_1), ..., \underline{\alpha}(t_n))\}$
$\underline{\alpha}(x) = \alpha(x)$,

  - *to each literal $c = p(t_1, .., t_n)$ in C, the set of assignments*

$$Sol_I(c) = \{\alpha : V \mapsto D_I \mid (\underline{\alpha}(t_1), ..., \underline{\alpha}(t_n)) \in p_I\}$$

where $\underline{\alpha}$ is defined as before.

Each assignment in $Sol_I(c)$ is a solution *of c in I.*

Example 1 *For instance, consider $F = \{o, succ\}$, $P = \{=\}$, and choose the natural numbers as the domain of interpretation. o is interpreted as 0 in natural numbers and succ as the successor operation. Moreover, $=$ is interpreted as the equality on natural numbers. The equation denoted $succ(succ(x)) =^? succ(y)$ is a constraint in C. The constraint $succ(succ(x)) =^? succ(y)$ of C is satisfiable since in this interpretation on natural numbers, it is possible to find a valuation of x and y, say n and $n + 1$ such that $succ(succ(x))$ and $succ(y)$ evaluate to equal natural numbers. $(x \mapsto n)(y \mapsto n + 1)$ is a solution.*

Now the language $ACL[F, P]$ can be enriched by adding conjunction, negation, existential quantification, to get a constrained language $SL[F, P]$.

Definition 3 *[Smo89] The* symbolic constraint language $SL[F,P]$ *is defined by:*

- *if* c, c' *are constraints in* $ACL[F,P]$, *then they are also constraints in* $SL[F,P]$, *and their* conjunction $c \wedge c'$, *the* trivial symbolic constraint $\top$, *the* negation $\neg c$, *the* existential quantification $\exists x.c$ *are in* $SL[F,P]$.

- *the variable mapping of* $ACL[F,P]$ *is extended as follows:*
  $\mathcal{V}(c \wedge c') = \mathcal{V}(c) \cup \mathcal{V}(c')$,
  $\mathcal{V}(\top) = \emptyset$,
  $\mathcal{V}(\neg c) = \mathcal{V}(c)$,
  $\mathcal{V}(\exists x.c) = \mathcal{V}(c) - \{x\}$.

- *If* I *is an interpretation of* $ACL[F,P]$, *the solution mapping of* I *is extended as follows:*
  $Sol_I(c \wedge c') = Sol_I(c) \cap Sol_I(c')$,
  $Sol_I(\top) = ASS_I$, *the set of all assignments from* V *to* I,
  $Sol_I(\neg c) = ASS_I - Sol_I(c)$,
  $Sol_I(\exists x.c) = \{\alpha \in ASS_I \mid$ *there exists* $\beta \in Sol_I(c), \beta = \alpha$ *on* $\mathcal{V}(c) - \{x\}\}$.

As usual the following abbreviations are used:

- $\neg(\top)$ is denoted $\bot$, defining the *unsatisfiable symbolic constraint*,

- $\neg(\neg c \wedge \neg c')$ is denoted $(c \vee c')$, so defining disjunction,

- $\neg(\exists x.\neg c)$ is denoted $(\forall x.c)$, defining universal quantification,

- $(\neg c) \vee c'$ is denoted $(c \Rightarrow c')$, defining implication.

The notations $\exists W$ and $\forall W$ are also used instead of $\exists x_1, ..., x_n$ and $\forall x_1, ..., x_n$ when $W = \{x_1, ..., x_n\}$.

2.2 Constraint subsumption and equivalence

Equivalence of constraints and renamings are crucial properties that should be cautiously defined.

A *renaming* is a bijective mapping $\rho : V \mapsto V$ equal to identity except on a finite number of variables in V. If c is a constraint, any $\rho(c)$ where ρ is a renaming, is called a *variant* of c.

Given a subset W of V, the W-solutions of a constraint c in an interpretation I is the set of solutions restricted to W: $Sol_I(c)_{|W} = \{\alpha_{|W} | \alpha \in Sol_I(c)\}$. A constraint c is *W-subsumed* by a constraint c', denoted $c \preceq_W c'$, if $Sol_I(c)_{|W} \subseteq Sol_I(c')_{|W}$. W-subsumption defines a quasi-ordering on constraints and induces an equivalence denoted $\simeq_W$: two constraints c and c' are *W-equivalent*, if $Sol_I(c)_{|W} = Sol_I(c')_{|W}$. For instance, any symbolic constraint c such that $Sol_I(c) = \emptyset$ is equivalent to the unsatisfiable symbolic constraint $\bot$. In all these definitions, W may be omitted if equal to V.

2.3 Some particular symbolic constraint languages

In what follows, we assume given, over the set of terms $\mathcal{T}(F,V)$, an equational theory T defined by a set of axioms also denoted by T. The interpretation domain we are interested in, is the set of ground terms $\mathcal{T}(F)$. We assume that the set of predicates P contains:

- the equality predicate, denoted $=$, interpreted either as the syntactic equality denoted $=_\emptyset$, or as the equality in equivalence classes, denoted $=_T$,
- the ordering predicate $\prec$ interpreted either as a complete simplification ordering $\prec_\emptyset$ on $\mathcal{T}(F)$, or as a reduction ordering $\prec_T$, compatible with T on equivalence classes modulo T.

The definitions of reduction and simplification orderings are reminded below.

Definition 4 *An ordering $\prec$ on $\mathcal{T}(F,V)$ is a* reduction ordering *if:*

1. *$\prec$ is well-founded,*
2. *$\forall s,t,w \in \mathcal{T}(F,V)$, $s \prec t$ implies $w[s] \prec w[t]$,*
3. *$\forall s,t \in \mathcal{T}(F,V)$, for any substitution σ, $s \prec t$ implies $\sigma(s) \prec \sigma(t)$.*

A reduction ordering $\prec$ is compatible *with T iff for any terms s,s',t,t', $s' =_T s \prec t =_T t'$ implies $s' \prec t'$.*

A total ordering $\prec$ on $\mathcal{T}(F) \bigcup \mathcal{A}(P)$ is a complete simplification ordering *if it is a reduction ordering and moreover:*

1. *for any proper subterm s of t, we have $s \prec t$,*
2. *for any atom A which contains s as a subterm, and any $t \prec s$, we have $s = t \prec A$, except when A is an equality $s = u$ with $t \prec u$; in that case $A \prec s = u$.*

Note that three problems have to be considered with respect to the equality and ordering predicates: their *validity* in the given (ground term) interpretation (every assignment has to be a solution), their *satisfiability* in the given interpretation (does there exist an assignment which is a solution?) and their solving in the given interpretation (enumerate or schematize the set of solutions). It is important to make a clear distinction between these problems; in order to emphasize the fact that, for constraints, we are essentially interested in the satisfiability and solving problems, we denote an atomic constraint built over the predicate P with a question mark: $P^?$. Thus we consider the following *elementary constraints*, where t and t' are terms:

1. $t =^?_\emptyset t'$, called an *equation*,
2. $\neg(t =^?_\emptyset t')$, denoted $(t \neq^?_\emptyset t')$ and called a *disequation*,
3. $t =^?_T t'$, called an *equation modulo T*,

4. $\neg(t =^?_T t')$, denoted $(t \neq^?_T t')$ and called a *disequation modulo* T.

5. $t \prec^?_\emptyset t'$, called an *inequation*,

6. $t \prec^?_T t'$, called an *inequation modulo* T.

For all these constraints, $\mathcal{V}(c) = \mathcal{V}(t) \cup \mathcal{V}(t')$.

In other words, we are interested in a constraint language denoted $SL_T[F, P]$, which has the two following instances:

1. $SL_\emptyset[F, P]$, where $P \supseteq \{=, \prec\}$, with $\mathcal{T}(F) \cup \mathcal{A}(P)$ as interpretation domain, and $=_\emptyset, \prec_\emptyset$ as interpretations of respectively $=$ and $\prec$.

2. $SL_T[F, \{=, \prec\}]$ with the quotient set of ground terms by the equality generated by T, denoted $\mathcal{T}(F)/=_T$, as interpretation domain, and $=_T, \prec_T$ as interpretations of respectively $=$ and $\prec$. We will especially consider the case where T is an associative commutative theory (AC for short).

Note that this can be extended in order to handle, in particular, typing constraint in the spirit of [Sch87, Kir88, Com90a].

In these interpretations, the solution mapping associates, to a constraint, a set of substitutions. $Sol(c)$ denotes in the following, the result of the solution mapping applied to c in an interpretation I that is a set of terms or a quotient set of terms. For a detailed exposition of these concepts see [JK90]. The following definitions are given for any theory T that may be the empty theory.

Definition 5 *A substitution σ is* W-subsumed modulo T *by a substitution σ', denoted $\sigma \leq^W_T \sigma'$ if there exists α such that, for any variable x in the set of variables W, $\alpha(\sigma(x)) =_T \sigma'(x)$. The substitution σ is said* more general *than σ' on W.*

W may be omitted if equal to V. We also write $\alpha =^W_T \beta$ if, for any variable x in the set of variables W, $\alpha(x) =_T \beta(x)$.

For the kind of constraints considered here, more general solutions can be defined and are assumed to be idempotent, without lost of generality. W-subsumption defines a quasi-ordering on substitutions and induces the following equivalence:

Definition 6 *Let W be a given set of variables. Two substitutions σ and σ' are* W-equivalent modulo T, *denoted $\sigma \simeq^W_T \sigma'$ if $\sigma \leq^W_T \sigma'$ and $\sigma' \leq^W_T \sigma$.*

This leads to the following definition of generating sets of solutions:

Definition 7 *A set of substitutions Σ is a* complete set of solutions of a constraint c *if*

- $\forall \sigma \in \Sigma$, $\sigma(c)$ *holds,*
- $\forall \alpha \in Sol(c), \exists \sigma \in \Sigma$ *such that* $\sigma \leq^{\mathcal{V}(c)}_T \alpha$.

A complete set of solutions of a constraint c is denoted by $CSS(c)$. Remind that when c is a conjunction of equations in the empty theory, a minimal complete set of solutions is either empty or reduced to one element called the *most general unifier* of c, and denoted by $mgu(c)$ in short.

2.4 Constraint solver

Two important concerns on symbolic constraints are first their satisfiability with respect to the considered class of interpretations, second their solving, that is the computation of the set of solutions of the constraints. Since this set of solutions is likely to be infinite, it is in general represented by a complete set of solutions, often computed from a specific form of the constraints called solved forms.

Various notions of solved forms have been introduced, in particular in the unification community [JK90]. In order to illustrate the concept of solved form, let us consider the case of combinations of equations and inequations, following [Com90b].

Definition 8 *A* solved system *is any conjunction of equations and inequations of the form*

$$\bigwedge_{i\in I}(x_i=^?_\emptyset t_i)\bigwedge_{j\in J}(x_j \prec^?_\emptyset t_j)\bigwedge_{k\in K}(t_k \prec^?_\emptyset x_k)$$

such that

1. *for all $i \in I$, x_i occurs once in the formula.*
2. *for any $n \in J \cup K$, x_n does not occur in t_n.*

A symbolic constraint c is said in solved form *if c is $\top$, $\bot$ or a disjunction of solved systems.*

Note that this definition forbids systems containing for instance $(x=^?_\emptyset f(y))$ and $(y=^?_\emptyset g(z))$, that are often said in dag-solved form [JK90]. The solved form would be $(x=^?_\emptyset f(g(z))) \wedge (y=^?_\emptyset g(z))$. This definition also forbids disequations: a constraint $(x \neq^?_\emptyset y)$ must be equivalent in the considered interpretation to a disjunction of $(x=^?_\emptyset t_i) \wedge (y=^?_\emptyset t'_i)$.

We assume given in the following a process, called a *constraint solver*, that for any symbolic constraint c, checks if c is satisfiable and possibly computes its solved form, denoted $c\downarrow$. Such a process does not exist in general. But for the kind of constraints we are considering in the examples, constraint solvers are already available.

Consider constraints that are disjunctions of systems (or conjunctions) only composed of:

- equations in the free term algebra: then a unification algorithm provides a constraint solver.

- equations in a quotient algebra by a set of axioms T: then a T-unification algorithm provides a constraint solver. The T-unification problem is in general undecidable, but specific T-unification algorithms do the job, when they exist. Moreover narrowing [Fay79, Hul80, JKK83] may also provide a constraint solver for theories with a convergent term rewrite system.

- inequations for a given simplification ordering, such as the lexicographic path ordering. This problem has recently been solved in [Com90b]. Is is developed in the next section 2.5.

- equations and disequations, with ground terms as interpretation domain. These questions are solved by [Com88, CL89]. For symbolic constraints composed only of equations and disequations without universal quantification in AC theories, these questions are also solved in [Com88]. However it is a challenging problem to get satisfiability and solving decision procedures for such symbolic constraints in other equational theories of interest.

2.5 Equality and inequality constraint solving

In order to illustrate a particular process of constraint solving used in Section 8, we focus in this section on equality and inequality constraint solving. Whenever it is assumed that $\prec$ is interpreted as a total ordering, inequality constraints do not have negation: this is not needed, since any constraint $\neg(x \prec^?_\emptyset y)$ is equivalent to $(y \prec^?_\emptyset x \vee x =^?_\emptyset y)$, that is $Sol(\neg(x \prec^?_\emptyset y)) = Sol(y \prec^?_\emptyset x \vee x =^?_\emptyset y)$.

The constraint solving rules given below depend from the definition of $\prec_\emptyset$. They are very natural extensions of rules for unification problems and similar rules have been presented for the first time in [Com90b].

A complete simplification ordering on $\mathcal{T}(F) \bigcup \mathcal{A}(P)$ is built as a variant of the *lexicographic path ordering*, which is known as a simplification ordering [Der87]:

Definition 9 *Let $<_F$ be a total ordering on F, called* precedence, *then ground terms are compared as follows:*

$$s = f(s_1, ..., s_n) \prec_{lpo} g(t_1, ..., t_m) = t \text{ if}$$

- *either* $\exists i \in 1 \ldots m$, $s \prec_{lpo} t_i$ *or* $s = t_i$
- *or* $f <_F g$ *and* $\forall i \in 1 \ldots n$, $s_i \prec_{lpo} t$
- *or* $f = g$, $(s_1, ..., s_n) \prec_{llpo} (t_1, ..., t_n)$ *and* $\forall j \in [1..n]$, $s_j \prec_{lpo} t$

where $\prec_{llpo}$ is the lexicographic extension of $\prec_{lpo}$.
Let $<_P$ be a total ordering on P, such that the equality predicate $=$ is minimal.
Ground atoms are compared as follows:

$$A = M(s_1, ..., s_n) \prec_{lpo} Q(t_1, ..., t_m) = B \text{ if}$$

- *either* $M <_P Q$

- *or* $M = Q$, M *is the equality predicate and* $\{s_1, s_2\} \prec_{mlpo} \{t_1, t_2\}$
- *or* $M = Q$, M *is not the equality predicate and* $(s_1, ..., s_n) \prec_{llpo} (t_1, ..., t_n)$

where $\prec_{mlpo}$ *is the multiset extension of* $\prec_{lpo}$.

In order to detect unsolvable constraints, we shall need, in the applications we have in mind, to test if a conjunction of inequations or equations is solvable. The following result about constraints on terms is easily extended to atoms.

Proposition 1 *[Com90b] It is decidable whether a formula* $s \preceq_{lpo} t$ *is solvable in* $\mathcal{T}(F)$.

Now, a few inference rules are introduced to solve constraints involving the ordering just defined. The equational part is the now usual presentation of unification algorithms, as pionered by Martelli-Montanari and G. Huet [Hue76, MM82]. The inequational part is very similar to [Com90b]. However the general strategy is too complex and only a subset of the complete set of the inference rules is given here. In the following rules $\vec{s}$ denotes the tuple $(s_1, \ldots, s_n)$.

Small $c \wedge f(s_1, \ldots, s_n) \prec^?_\emptyset g(t_1, \ldots, t_m)$

$\mapsto\!\!\!\!\rightarrow$

$c \wedge \bigwedge_{i=1}^{n}(s_i \prec^?_\emptyset g(t_1, \ldots, t_m))$

if $f <_F g$

Big $c \wedge f(s_1, \ldots, s_n) \prec^?_\emptyset g(t_1, \ldots, t_m)$

$\mapsto\!\!\!\!\rightarrow$

$c \wedge \bigvee_{i=1}^{m}(f(s_1, \ldots, s_n) \prec^?_\emptyset t_i)$

if $g <_F f$

Same $c \wedge f(s_1, \ldots, s_n) \prec^?_\emptyset f(t_1, \ldots, t_n)$

$\mapsto\!\!\!\!\rightarrow$

$c \wedge \bigwedge_{i=1}^{n}(s_i \prec^?_\emptyset f(t_1, \ldots, t_n))$

$\wedge((s_1 \prec^?_\emptyset t_1) \vee (s_1 =^?_\emptyset t_1 \wedge s_2 \prec^?_\emptyset t_2) \vee .. \vee (s_1 =^?_\emptyset t_1 \wedge \cdots \wedge s_n \prec^?_\emptyset t_n))$

SmallP $c \wedge M(s_1, \ldots, s_n) \prec^?_\emptyset Q(t_1, \ldots, t_m)$

$\mapsto\!\!\!\!\rightarrow$

c

if $M <_P Q$

BigP $c \wedge M(s_1, \ldots, s_n) \prec^?_\emptyset Q(t_1, \ldots, t_m)$

$\mapsto\!\!\!\!\rightarrow$

$\bot$

if $Q <_P M$

SameP $c \wedge Q(s_1, \ldots, s_n) \prec^?_\emptyset Q(t_1, \ldots, t_n)$

$\mapsto\!\!\!\!\rightarrow$

$c \wedge \bigwedge_{i=1}^{n}(s_i \prec^?_\emptyset Q(t_1, \ldots, t_n))$

$\wedge((s_1 \prec^?_\emptyset t_1) \vee (s_1 =^?_\emptyset t_1 \wedge s_2 \prec^?_\emptyset t_2) \vee \cdots \vee (s_1 =^?_\emptyset t_1 \wedge .. \wedge s_n \prec^?_\emptyset t_n))$

if $Q \neq =^?_\emptyset$

SameE $c \wedge (s_1 =^?_\emptyset s_2) \prec^?_\emptyset (t_1 =^?_\emptyset t_2)$

$\mapsto\!\!\!\!\rightarrow$

$c \wedge ((s_1 =^?_\emptyset t_1 \wedge s_2 \prec^?_\emptyset t_2) \vee (s_1 =^?_\emptyset t_2 \wedge s_2 \prec^?_\emptyset t_1)$

$\vee (s_2 =^?_\emptyset t_1 \wedge s_1 \prec^?_\emptyset t_2) \vee (s_2 =^?_\emptyset t_2 \wedge s_1 \prec^?_\emptyset t_1)$

$\vee (s_1 \prec^?_\emptyset t_1 \wedge s_2 \prec^?_\emptyset t_1) \vee (s_1 \prec^?_\emptyset t_2 \wedge s_2 \prec^?_\emptyset t_2))$

Path ordering rules

Delete	$c \wedge s =^?_\emptyset s$	$\mapsto\!\!\!\!\rightarrow$	c
Decompose	$c \wedge f(\vec{s}) =^?_\emptyset f(\vec{t})$	$\mapsto\!\!\!\!\rightarrow$	$c \wedge s_1 =^?_\emptyset t_1 \wedge \cdots \wedge s_n =^?_\emptyset t_n$
Conflict	$c \wedge f(\vec{s}) =^?_\emptyset g(\vec{t})$	$\mapsto\!\!\!\!\rightarrow$	$\bot$ if $f \neq g$
Check	$c \wedge x =^?_\emptyset s[x]$	$\mapsto\!\!\!\!\rightarrow$	$\bot$
Coalesce	$c \wedge x =^?_\emptyset y$	$\mapsto\!\!\!\!\rightarrow$	$\{x \mapsto y\}c \wedge x =^?_\emptyset y$ if $x, y \in \mathcal{V}(c)$ *and* $x \neq y$
Replace	$c \wedge x =^?_\emptyset s$	$\mapsto\!\!\!\!\rightarrow$	$\{x \mapsto s\}c \wedge x =^?_\emptyset s$ if $x \notin \mathcal{V}(s), s \notin V$ *and* $x \in \mathcal{V}(c)$

Equational rules

Subterm	$c \wedge t \prec^?_\emptyset s[t]$	$\mapsto\!\!\!\!\rightarrow$	c
Superterm	$c \wedge s[t] \prec^?_\emptyset t$	$\mapsto\!\!\!\!\rightarrow$	$\bot$
Transitivity	$c \wedge s \prec^?_\emptyset t \wedge t \prec^?_\emptyset s$	$\mapsto\!\!\!\!\rightarrow$	$\bot$

Simplification ordering rules

3 Constrained formulae

The other component of the syntax for constrained deduction is first-order formulae. Several kinds of constrained formulae are considered here: equalities, rewrite rules and clauses. We also make precise their respective semantics. Let Σ be a signature of function symbols F and predicate symbols P. We recall that a *first-order Σ-structure* $\mathcal{M}$ is given by a carrier set, a function for each function symbol in F and a relation for each relation symbol in P. The Σ-structure $\mathcal{M}$ is *term-generated* if each element of the carrier of $\mathcal{M}$ is the interpretation of a term in $\mathcal{T}(F)$. Whenever Σ contains the predicate symbol $=$, it will be interpreted as the equality relation in $\mathcal{M}$. Given a set of axioms Th, a model of Th is a Σ-structure which satisfies Th. Whatever formulae we shall consider, we shall mainly be interested by its term-generated models. In particular, constrained equational theories admit an initial model which is term-generated, since it can be obtained as a quotient of the ground terms algebra by a congruence.

3.1 Constrained equalities and rewrite rules

Definition 10 *A* constrained equality, *denoted* $(l = r, c)$, *is given by two terms* l *and* r *in* $\mathcal{T}(F, X)$ *and a constraint* c *in* $SL_{\mathcal{T}}[F, P]$.

Then a constrained equality $(l = r, c)$ schematizes the following set of equalities:

$$\mathcal{S}(l = r, c) = \{\sigma(l) = \sigma(r) | \sigma \in Sol(c)\}.$$

Constrained equalities may be oriented to produce constrained rules.

Definition 11 *A* constrained rewrite rule, *denoted* $(l \rightarrow r, c)$, *is given by two ordered terms* l, r *in* $\mathcal{T}(F, X)$ *and a constraint* c *in* $SL_T[F, P]$.

A constrained rewrite rule $(l \rightarrow r, c)$ schematizes the following set of rewrite rules:

$$\mathcal{S}(l \rightarrow r, c) = \{\sigma(l) \rightarrow \sigma(r) | \sigma \in Sol(c)\}.$$

Definition 12 *A* constrained equational theory *is given by the constraint language* $SL_T[F, P]$ *and a set of constrained equalities* E *built on* $\mathcal{T}(F, X)$.

The model of a constrained equational theory considered here, is an initial algebra built as a quotient algebra by the set of equalities schematized by E. More formally,

Definition 13 *Let a constrained theory be given by the constraint language* $SL_T[F, P]$ *and a set of constrained equalities* E. *Let us define*

$$\mathcal{S}(E) = \bigcup_{(l=r,c) \in E} \mathcal{S}(l = r, c)$$

and let $=_{\mathcal{S}(E) \cup T}$ *the smallest congruence generated by* T *and by this (possibly infinite) set of equalities on* $\mathcal{T}(F)$. *The* initial model of the constrained theory $(SL_T[F, P], E)$ *is the initial quotient algebra* $\mathcal{T}(F)/ =_{\mathcal{S}(E) \cup T}$. *Its* term-generated models *are the term-generated models of* $\mathcal{S}(E) \cup T$.

Finding a set of constrained rewrite rules R *equivalent* to a set of constrained equalities E needs to make precise the notion of equivalence: let us define

$$\mathcal{S}(R) = \bigcup_{(l \rightarrow r, c) \in R} \mathcal{S}(l \rightarrow r, c).$$

R is *equivalent* to E if $=_{\mathcal{S}(E) \cup T}$ and $=_{\mathcal{S}(R) \cup T}$ coincide on $\mathcal{T}(F)$, where $=_{\mathcal{S}(R) \cup T}$ is the smallest congruence generated by T and by this possibly infinite set of rewrite rules on $\mathcal{T}(F)$.

3.2 Constrained clauses

Definition 14 *A* constrained clause, *denoted* (C, c), *is given by a first-order clause* $C = L_1 \vee L_2 \vee \cdots \vee L_n$ *(that may contain equalities) and a constraint* c *in* $SL_T[F, P]$.

A constrained clause (C, c) schematizes the following set of clauses:

$$\mathcal{S}(C, c) = \{\sigma(C) | \sigma \in Sol(c)\}.$$

This is the set of all the ground instances $\sigma(C)$ of C, where the substitution σ belongs to the set of solutions of c.

A model of a constrained clause is a term-generated model for the set of its ground instances.

Definition 15 *A* constrained first-order theory *is given by the constraint language* $SL_T[F, P]$ *and a set* L *of constrained clauses. Its* term-generated models *are the term-generated models of* $\mathcal{S}(L) \cup T$, *where*

$$\mathcal{S}(L) = \bigcup_{(C,c)\in L} \mathcal{S}(C, c).$$

In the following, we speak of a *constrained formula* (F, c) to denote any of the previously introduced constrained equality, rewrite rule or clause. We identify a first-order formula F with a formula constrained by the trivial problem $\top$, namely $(F, \top)$.

In the considered sets of constrained formulae, we shall assume that their sets of variables are disjoint, without lost of generality. Two constrained formulae (F, c) and (F', c') are *variable disjoint* if $\mathcal{V}(F) \cup \mathcal{V}(c)$ and $\mathcal{V}(F') \cup \mathcal{V}(c')$ have no common element.

4 Constrained deduction rules for equational logic

We first focus on equational logic and present in this section constrained versions of deduction rules on formulae built with terms and the symbols $=$ or $\rightarrow$.

4.1 Constrained superposition

A general definition of superposition of a constrained rewrite rule on a constrained formula is given.

Definition 16 *The constrained rewrite rule* $(l \rightarrow r, c)$ *overlaps modulo* T *the constrained formula* (F, c') *at occurrence* m *in* F *if the constraint* $(c \wedge c' \wedge (F_{|m} =^?_T l))$ *is satisfiable. The* deduced constrained formula modulo T *obtained by superposition of* $(l \rightarrow r, c)$ *into* (F, c') *at occurrence* m *is by definition:*

$$(F[r]_m, c \wedge c' \wedge (F_{|m} =^?_T l)).$$

It should be noted that in general superposition at non-variable occurrences are only considered.

A deduced constrained formula of the form

$$(F[r]_m, (c \wedge c' \wedge (F_{|m} =^?_T l)))$$

schematizes the following set of formulae:

$$\mathcal{CP} = \{\sigma(F[r]_m) | \sigma \in Sol(c \wedge c' \wedge (F_{|m} =^?_T l))\}.$$

The constrained version of the superposition rule is the following:

Deduce $CF \cup \{(F, c'), (l \to r, c)\}$
$\mapsto\!\!\!\!\to$
$CF \cup \{(F, c'), (l \to r, c), (F[r]_m, c \wedge c' \wedge (F_{|m} =^?_T l))\}$
if $c \wedge c' \wedge (F_{|m} =^?_T l)$ is satisfiable

Superposition Rule

Proposition 2 *The term-generated models of* $(F, c') \cup (l \to r, c)$ *and* $(F[r]_m, c \wedge c' \wedge (F_{|m} =^?_T l)) \cup (F, c') \cup (l \to r, c)$ *are the same.*

Proof: Let $E_1 = \mathcal{S}(F, c') \cup \mathcal{S}(l \to r, c) \cup T$ and $E_2 = \mathcal{S}(F[r]_m, (c \wedge c' \wedge (F_{|m} =^?_T l))) \cup \mathcal{S}(F, c') \cup \mathcal{S}(l \to r, c) \cup T$.

It is sufficient to show that each element of each set is a logical consequence of the elements of the other set. The only point to prove is that an element e of $\mathcal{S}(c \wedge c' \wedge (F_{|m} =^?_T l))$ is entailed by E_1. There is a solution σ of $(c \wedge c' \wedge (F_{|m} =^?_T l)$ such that $e = \sigma(F[r]_m)$. But $\sigma(l) =_T F_{|m}$, $\sigma(l) = \sigma(r) \in E_1$ and $\sigma(F) \in E_1$. Therefore, by equational reasoning on these formulae, we can derive e. □

When (F, c) is a constrained rewrite rule and m is an occurrence of its left-hand side, we obtain the concept of *constrained critical pair* that schematizes critical pairs used in completion modulo T.

Example 2 *Consider the two rewrite rules*

$$x * x * x * x \to x$$
$$u * v * w * a \to a.$$

where $*$ *is supposed to be AC (associative and commutative). By constrained superposition at top occurrence, the constrained critical pair*

$$(\langle x, a \rangle,\ (x * x * x * x =^?_{AC} u * v * w * a))$$

is computed. It schematizes for instance the trivial critical pair $\langle a, a \rangle$ *obtained with the solution* $\{(x \mapsto a)(u \mapsto a)(v \mapsto a)(w \mapsto a)\}$*; it also schematizes the critical pair* $\langle a * x', a \rangle$*, by considering the solution* $\{(x \mapsto a * x')(u \mapsto a * x' * x')(v \mapsto a * x')(w \mapsto a * x')\}$.

4.2 Constrained simplification

We consider in this section several forms of constrained simplification. To simplify a constrained formula (F_1, c_1) using a constrained rewrite rule $(l \to r, c)$, we assume that the two constrained formulae are variable disjoint. This condition can always be satisfied by renaming variables of the constrained rewrite rule.

Simplification of constrained formulae is more complex than constrained deduction. This is because when simplifying, the starting formula is *replaced* by the simplified one and lost, while otherwise, when a deduction step is applied, a new formula is *added*. In the simplification process, it must be taken care of not loosing schematized instances of the initial formula. This is why the definition of simplification involves two parts. The first one deals with instances which are really simplified by an instance of the constrained rewrite rule. The second part deals with the non-simplifiable instances and records the failure in the constraint.

Definition 17 *A constrained formula* (F_1, c_1) simplifies modulo T *to* (F_2, c_2), *with the rest* (F_1, c_2'), *at the non-variable occurrence* m *of* F_1, *with the constrained rewrite rule* $(l \to r, c)$, *if the constraint* $c_2 = c_1 \wedge (c \wedge (l =^?_T F_{1|m}))$ *is satisfiable. Then* $F_2 = F_1[r]_m$ *and* $c_2' = (c_1 \wedge [\forall \mathcal{V}(c) \cup \mathcal{V}(l).\neg c \vee \neg(l =^?_T F_{1|m})])$. *This is denoted by* $(F_1, c_1) \rightarrowtail^T \{(F_2, c_2), (F_1, c_2')\}$.

The notion of constrained rewriting takes its interest from the fact that checking satisfiability of a constraint is in general much simpler than finding a complete set of solutions or a solved form, especially in equational theories. For instance, a criterion to check the satisfiability of a system of equations modulo associativity-commutativity is given in [vZGS78]. Finding complete sets of AC-solutions of equation systems is much more difficult.

The previous definition given for simplifying a constrained formula (F, c) takes into account all the schematized formulae. Some of these instances that are not reducible are recorded in the rest. When the constraint associated to the rest is unsatisfiable, this means that all instances of the formula are reducible. We call this case a total simplification. It is possible to restrict simplification with this weaker definition.

Definition 18 *A constrained formula* (F_1, c_1) totally simplifies modulo T *to* (F_2, c_2), *at the non-variable occurrence* m *of* F_1, *with the constrained rewrite rule* $(l \to r, c)$, *if* $c_2 = c_1 \wedge (c \wedge (l =^?_T F_{1|m}))$ *is satisfiable and if* $c_2' = (c_1 \wedge [\forall \mathcal{V}(c) \cup \mathcal{V}(l).\neg c \vee \neg(l =^?_T F_{1|m})])$ *is unsatisfiable. This is denoted by* $(F_1, c_1) \rightarrowtail\!\!\!\!\rightarrow^T (F_2, c_2)$.

To summarize, two deduction rules for simplification of constrained formulae can be given:

Simplify	$CF \cup \{(F_1, c_1)\}$	$\mapsto\!\!\!\!\rightarrow$	$CF \cup \{(F_2, c_2), (F_1, c_2')\}$ if $(F_1, c_1) \rightarrowtail^T \{(F_2, c_2), (F_1, c_2')\}$
TotallySimplify	$CF \cup \{(F_1, c_1)\}$	$\mapsto\!\!\!\!\rightarrow$	$CF \cup \{(F_2, c_2)\}$ if $(F_1, c_1) \rightarrowtail\!\!\!\!\rightarrow^T (F_2, c_2)$

Simplification Rules

Let us first address the correctness problem. From the point of view of schematization, the set of constrained formulae $\{(F_2, c_2), (F_1, c_2')\}$ schematizes

$\mathcal{S}(F_2, c_2) \cup \mathcal{S}(F_1, c_2')$. The models are preserved by the simplification rule, thanks to the following proposition:

Proposition 3 *The term generated models of* $(F_2, c_2) \cup (F_1, c_2') \cup (l \to r, c) \cup T$ *and* $(F_1, c_1) \cup (l \to r, c) \cup T$ *are the same.*

Proof: Let $E_1 = \mathcal{S}(F_2, c_2) \cup \mathcal{S}(F_1, c_2') \cup \mathcal{S}(l \to r, c) \cup T$ and $E_2 = \mathcal{S}(F_1, c_1) \cup \mathcal{S}(l \to r, c) \cup T$.

It is sufficient to show that each element of each set is a logical consequence of the elements of the other set. Let us prove that every element of E_2 is entailed by E_1. Consider for instance an element e of $\mathcal{S}(F_1, c_1)$. There is a solution σ of c_1 such that $e = \sigma(F_1)$. If σ satisfies c_2' then e is an element of $\mathcal{S}(F_1, c_2')$. Otherwise we have $\sigma(l) =_T F_{1|m}$, $\sigma(l) = \sigma(r) \in E_1$ and $\sigma(F_2) \in E_1$. Therefore, by equational reasoning on these formulae, we can derive e. We can prove in a similar fashion that E_1 is entailed by E_2.

□

In the case of a total simplification, the term-generated models of $\mathcal{S}(F_2, c_2) \cup \mathcal{S}(l \to r, c) \cup T$ and $\mathcal{S}(F_1, c_1) \cup \mathcal{S}(l \to r, c) \cup T$ are the same.

Example 3 *Consider the constrained rewrite rule,*

$$(x * x * x * x \to x, (x \neq_\emptyset^? a))$$

where $*$ *is denoted in infix notation and the expressions are supposed to be left parenthesized, whenever parentheses are implicit. The constrained equality* $(b * b * b * b) * (y * y * y * y) = a, \top)$ *with the trivial constraint* $\top$ *simplifies to another constrained equality*

$$(b * b * b * b * x = a,\ (x * x * x * x =_\emptyset^? y * y * y * y) \wedge (x \neq_\emptyset^? a))$$

with the rest

$$(b * b * b * b) * (y * y * y * y) = a,\ \forall x. \neg(x * x * x * x =_\emptyset^? y * y * y * y) \vee \neg(x \neq_\emptyset^? a)))$$

that reduces to

$$(b * b * b * b) * (y * y * y * y) = a,\ (y =_\emptyset^? a)).$$

The first branch simplifies then to $(x' * x = a, c)$ *with* c *being*

$$\begin{array}{c} (x * x * x * x =_\emptyset^? y * y * y * y) \wedge (x \neq_\emptyset^? a) \\ \wedge \\ (x' * x' * x' * x' =_\emptyset^? b * b * b * b) \wedge (x' \neq_\emptyset^? a). \end{array}$$

with the rest

$$(b * b * b * b * x = a,\ (x * x * x * x =_\emptyset^? y * y * y * y) \wedge (x \neq_\emptyset^? a)) \wedge c''\}$$

where c'' is derived from c. More precisely, c'' is

$$\forall x'.(x' * x' * x' * x' \neq^?_\emptyset b * b * b * b) \vee (x' =^?_\emptyset a).$$

that reduces to $\bot$. So actually this last simplification step is a total simplification step.

Note that the substitution $(x \mapsto y)(x' \mapsto b)$ is solution of the constraint c. By using this instantiation , we get the usual rewriting derivation:

$$(b * b * b * b) * (y * y * y * y) \rightarrow b * b * b * b * y \rightarrow b * y.$$

Beyond correctness, the definition of such a simplification relation rises two other problems: termination and relationship with classical definitions of simplification.

Constrained simplification needs to check first that the list of constraints c_2 is satisfiable. Otherwise obviously termination problems arise. But this is not enough to ensure termination of the process, as shown in the next example. The growth of the constraints must be controlled too and this may be enabled by the reduction strategy.

Example 4 *Let us consider $T = \emptyset$, the constrained rewrite rule $(f(x) \rightarrow f(y), y \prec^?_\emptyset x)$ and the equality $((f(a) = b), \top)$. This equality is simplified into: $((f(y) = b), (x =^?_\emptyset a) \wedge (y \prec^?_\emptyset x))$ and $((f(a) = b), (\forall x, y.(x \neq^?_\emptyset a) \vee \neg(y \prec^?_\emptyset x))$. The first constrained equality $((f(y) = b), (x =^?_\emptyset a) \wedge (y \prec^?_\emptyset x))$ can be simplified again; for instance, we obtain after n steps:*

$$(f(y_n) = b, (x =^?_\emptyset a) \wedge y_n \prec^?_\emptyset \cdots \prec^?_\emptyset y \prec^?_\emptyset a).$$

Let us consider now relations with classical simplification rules, based on matching, which means that the variables from the target formula are not instantiated. The simplification proposed here, is a more general process, since it splits the ground instances of a formula between the classically simplifiable instances and the other ones. Hence, during this splitting process, the variables of the target formula can be eventually instantiated. If constraints are eagerly solved, then the algorithm to be used there is unification instead of matching. However, not to be mistaken, it is different from the superposition rule since the original formula is *deleted and replaced* by its simplified version.

We show now that our definition of constrained simplification generalizes former notions of simplification. More precisely, adding hypotheses on the kind of constraints and (or) on the way to solve them, gives back the classical definitions.

Let us examine how the simplification rule specializes when dealing with the empty theory. In the classical use of simplification, the replacement is performed only when $F_{1|m}$ is an instance $\sigma(l)$ of l. We can then write c'_2 as:

$$c'_3 = c_1 \wedge \forall \mathcal{V}(c) \cup \mathcal{V}(l).\ \neg\sigma(c) \vee\ \neg(l =^?_\emptyset \sigma(l)).$$

Assuming that $\mathcal{V}(l) \cap \mathcal{V}(\sigma(x)) = \emptyset$ for any variable x of l, the matching substitution σ allows eliminating all the variables of l. We get the following constraint equivalent to c'_3:

$$c'_4 = c_1 \wedge \forall \mathcal{V}(c) - \mathcal{V}(l).\ \neg\sigma(c).$$

Assuming, as in simplifying conditional term rewriting systems, that $\mathcal{V}(c) \subseteq \mathcal{V}(l)$ yields to

$$c''_4 = c_1 \wedge\ \neg\sigma(c).$$

Restricting the situation a little bit more and considering that c is an inequality constraint $r \prec^?_\emptyset l$ where $\mathcal{V}(r) \subseteq \mathcal{V}(l)$, the constraint can be simplified further in:

$$c'_5 = c_1 \wedge \neg\sigma(r \prec^?_\emptyset l)$$

Finally, assuming that $\prec^?_\emptyset$ is a total ordering, we obtain:

$$c'_6 = c_1 \wedge (\sigma(l \prec^?_\emptyset r) \vee \sigma(l =^?_\emptyset r)))$$

If we replace c'_2 by c'_6 in the Definition 17, we meet the definition of simplification used in [Pet90].

The total simplification rule can be specialized too when constraints are formulas from the empty theory and when it is applied after computing a matching substitution σ. When total simplification is allowed, we see that c_1 is equivalent by dichotomy to $c_1 \wedge ((\exists \mathcal{V}(c) - \mathcal{V}(l).\sigma(c)) \vee (\forall \mathcal{V}(c) - \mathcal{V}(l).\ \neg\sigma(c)))$ and since c'_4 is unsatisfiable, this is also equivalent to: $c_1 \wedge (\exists \mathcal{V}(c) - \mathcal{V}(l).\sigma(c))$. Note that the last expression and $(c_1 \wedge \sigma(c))$ have the same solutions when we restrict to the variables of c_1. Hence we can replace c_2 by c_1 in the definition of total simplification. This will be performed in Section 8.

4.3 Constrained deletion

A constrained equality $(p = q, c)$ can be deleted if for any solution σ of c, $\sigma(p) =_T \sigma(q)$. Which is negated by: there exists a solution σ of c such that $\sigma(p) \neq^?_T \sigma(q)$. So the constrained equality $(p = q, c)$ is deleted whenever the constraint $c \wedge (p \neq^?_T q)$ has no solution. This can also be written as an implication $(c \Rightarrow p =_T q)$.

Delete	$CF \cup \{(p = q, c)\}$	$\mapsto\!\!\!\!\mapsto$	CF if $(c \Rightarrow p =_T q)$

Deletion rule

It is especially important to notice that the **Delete** rule is more powerful than the classical rule for eliminating trivial equations. Consider for instance the constrained formula $(s(x) = s(s(x)), x \neq^?_\emptyset 0 \wedge (\forall y.\ x \neq^?_\emptyset s(y)))$. If the language only contains s and 0 as function symbols, then **Delete** can be applied. The correctness of **Delete** is stated again with respect to term-generated models:

Proposition 4 *Applying* **Delete** *to a set of formulae gives a set of formulae with the same term-generated models.*

Proof: This is an immediate consequence of the fact that, if σ is solution of c, $\sigma(p) =_T \sigma(q)$. □

4.4 Constrained orientation

Turning a constrained equality $(p = q, c)$ into a constrained rule is now made precise. Assume given a reduction ordering $\prec_T$ on terms, compatible with T. Since it is closed under substitution, if $p \prec_T q$ then for all $\sigma \in Sol(c)$, $\sigma(p) \prec_T \sigma(q)$. The constrained rule $(p \to q, c)$ can be added. Similarly $(q \to p, c)$ holds if $q \prec_T p$. However it may happen that some instances of $p = q$ satisfy $\sigma(p) \prec_T \sigma(q)$, while others satisfy $\sigma(q) \prec_T \sigma(p)$ or $\sigma(p) =_T \sigma(q)$.

The possible schematized instances of $(p = q, c)$ are split into three families. More formally, let us define the following constraints:

$$c'_1 = c \wedge (p \prec_T^? q)$$
$$c'_2 = c \wedge (q \prec_T^? p)$$
$$c'_3 = c \wedge \neg((p \prec_T^? q) \vee (q \prec_T^? p))$$

Then the orientation rule first checks that c'_1 and c'_2 are satisfiable. If only one, say c'_1 is satisfiable, then the constrained rule $(p \to q, c'_1)$ is added, and symmetrically if c'_2 only is satisfiable. If both are satisfiable, then $(p \to q, c'_1)$ and $(q \to p, c'_2)$ are added. Finally if c'_3 is satisfiable, the constrained equality $(p = q, c'_3)$ must subsist.

Orient $CF \cup \{(p = q, c)\}$
$\mapsto\!\!\!\!\to$
$CF \cup \{(p = q, c \wedge \neg((p \prec_T^? q) \vee (q \prec_T^? p))),$
$(p \to q, (c \wedge (q \prec_T^? p))), (q \to p, (c \wedge (p \prec_T^? q)))\}$

Orientation rule

Whenever $\prec_T$ can be extended to a total ordering on equivalence classes of ground terms, and when interested in ground completion, we get rid of the last set of constraints c'_3:

TotallyOrient $CF \cup \{(p = q, c)\}$
$\mapsto\!\!\!\!\to$
$CF \cup \{(p \to q, (c \wedge q \prec_T^? p)), (q \to p, (c \wedge p \prec_T^? q))\}$

Orientation rule with a total ordering

Proposition 5 *The term-generated models of* $(p = q, c)$ *and* $(p \to q,\ c \wedge (q \prec_T^? p)) \cup (q \to p,\ c \wedge (p \prec_T^? q)) \cup (p = q, c \wedge \neg((p \prec_T^? q) \vee (q \prec_T^? p)))$ *are the same.*

Proof: For any $\sigma \in Sol(c)$, one of the three constraints c'_1, c'_2 or c'_3 is satisfied.

Conversely, any solution of c'_1, c'_2 or c'_3 is also solution of c. □

5 Constrained deduction rules for first-order logic with equality

In first-order logic, by Herbrand's theorem, a universal formula is satisfiable if and only if it is the case for the set of its ground instances. Therefore, it is quite natural to consider a clause (and more generally a universal formula) as schematizing the set of all its ground instances. Inference rules for first-order logic can be adapted accordingly with the help of constraints. In the context of clausal theorem proving with equality, a complete system of inference rules can be obtained from *factoring*, *resolution* and *paramodulation*. These rules have been refined in [HR86] by imposing ordering conditions on the subterms where inferences are applied. These conditions are expressed with respect to a complete simplification ordering. We present here the constrained inference system derived from the one in [Rus88] by including the ordering and unifiability requirements within the constraints.

Each constrained inference rule generalizes the usual one in a straightforward way.

Definition 19

- *The* constrained factor *of the clause* $(L_0 \vee L_1 \vee \cdots \vee L_n, c)$ *is* $(L_1 \vee L_2 \vee \cdots \vee L_n, c')$ *if the constraint* $c' = c \wedge (L_0 =^?_\emptyset L_1) \wedge (\bigwedge_{i=2}^n L_i \preceq^?_\emptyset L_0)$ *is satisfiable.*
- *The* constrained resolvent *of the clauses* $(L_0 \vee L_1 \vee \cdots \vee L_n, c)$ *and* $(\neg L'_0 \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')$ *is* $(L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c'')$ *if the constraint*

$$c'' = c \wedge c' \wedge (L_0 =^?_\emptyset L'_0) \wedge (\bigwedge_{i=1}^n L_i \prec^?_\emptyset L_0) \wedge (\bigwedge_{i=1}^m L'_i \prec^?_\emptyset L'_0)$$

  is satisfiable.
- *The* constrained paramodulant *of the clause* $(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c)$ *into the clause* $(L'[u] \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')$ *is* $(L'[t] \vee L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c'')$ *if*

$$c'' = c \wedge c' \wedge (s =^?_\emptyset u) \wedge (t \prec^?_\emptyset s) \wedge (\bigwedge_{i=1}^m L'_i \prec^?_\emptyset L'[u])$$

  is satisfiable and $L'[u]$ *is not a positive equational literal.*

- *The* constrained superposition *of the clause* $(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c)$ *into the clause* $(a[u] = b \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')$ *is* $(a[t] = b \vee L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c'')$ *if*

$$c'' = c \wedge c' \wedge (s =^?_\emptyset u) \wedge (t \prec^?_\emptyset s) \wedge (b \prec^?_\emptyset a[u]) \wedge (\bigwedge_{i=1}^{m} L'_i \prec^?_\emptyset a[u] = b)$$

is satisfiable.

OFactoring
$CF \cup \{(L_0 \vee L_1 \vee \cdots \vee L_n, c)\}$
$\mapsto$
$CF \cup \{(L_0 \vee L_1 \vee \cdots \vee L_n, c)\} \cup \{(L_1 \vee L_2 \vee \cdots \vee L_n, c')\}$
if $c' = c \wedge (L_0 =^?_\emptyset L_1) \wedge (\bigwedge_{i=2}^{n} L_i \preceq^?_\emptyset L_0)$
OResolution
$CF \cup \{(L_0 \vee L_1 \vee \cdots \vee L_n, c), (\neg L'_0 \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$
$\mapsto$
$CF \cup \{(L_0 \vee L_1 \vee \cdots \vee L_n, c), (\neg L'_0 \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$
$\cup \{(L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c'')\}$
if $c'' = c \wedge c' \wedge (L_0 =^?_\emptyset L'_0) \wedge (\bigwedge_{i=1}^{n} L_i \prec^?_\emptyset L_0) \wedge (\bigwedge_{i=1}^{m} L'_i \prec^?_\emptyset L'_0)$
OParamodulation
$CF \cup \{(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c), (L'[u] \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$
$\mapsto$
$CF \cup \{(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c), (L'[u] \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$
$\cup \{(L'[t] \vee L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c'')\}$
if $c'' = c \wedge c' \wedge (s =^?_\emptyset u) \wedge (\bigwedge_{i=1}^{m} L'_i \prec^?_\emptyset L'[u]) \wedge (t \prec^?_\emptyset s)$
OSuperposition
$CF \cup \{(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c), (a[u] = b \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$
$\mapsto$
$CF \cup \{(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c), (a[u] = b \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$
$\cup \{(a[t] = b \vee L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c'')\}$
if $c'' = c \wedge c' \wedge (s =^?_\emptyset u) \wedge (t \prec^?_\emptyset s) \wedge (b \prec^?_\emptyset a[u]) \wedge (\bigwedge_{i=1}^{m} L'_i \prec^?_\emptyset a[u] = b)$

Constrained inference rules for first-order logic

Proposition 6 *Applying the inference rules of* **OFactoring**, **OResolution**, **OParamodulation** *and* **OSuperposition** *to a set of constrained clauses preserves its term-generated models.*

Proof: This proposition follows from the fact that the rules can be viewed merely as restriction of sound first-order deduction rules to ground clauses. □

The next example illustrates the fact that keeping track of the constraints during deduction can drastically reduce the number of inferences. The point is that any deduction step can be blocked whenever it generates unsolvable constraints.

Example 5 *Let us consider the following clause, where* $F = \{0, s, g\}$ *and* $P = \{Q\}$. *We define* $\prec_\emptyset^?$ *as the lexicographic rpo* $\prec_{lpo}$ *with the precedence:* $0 <_F s <_F g$.

$$\neg\, Q(g(x,y)) \vee Q(g(s(x),x))$$

Let us resolve this clause with itself. We first introduce a renamed version of the clause:

$$\neg\, Q(g(x',y')) \vee Q(g(s(x'),x'))$$

The constraint to be checked for satisfiability is:

$$Q(g(s(x),x)) \prec_{lpo} Q(g(x,y)) \wedge Q(g(x',y')) \prec_{lpo} Q(g(s(x'),x'))) \wedge$$
$$Q(g(s(x'),x'))) =_\emptyset^? Q(g(x,y))$$

Applying some of the rules given in Section 2.4 for solving inequational constraints, we get:

$$(s(s(x')),s(x')) \prec_{llpo} (s(x'),x') \wedge (x',y') \prec_{llpo} (s(x'),x') \wedge x =_\emptyset^? s(x') \wedge y =_\emptyset^? x'$$

which obviously admits no solution.

Note that applying classical resolution generates an infinite number of other clauses, among which:

$$\neg\, Q(g(x,y)) \vee Q(g(s^n(x), s^{n-1}(x)))$$

where $s^n(x)$ *abbreviates* $\underbrace{s(s(\cdots))}_{n}(x)$

6 Restructuring rules

Until now, our logical system works with two separated parts: the constraint part and the classical first-order part. We introduce now some rules which allow to transfer information from the constraints to the first-order part.

The first rule, called **Elim Or**, expresses that we can replace a set of instances of a clause by two subsets, when the constraint is a disjunction. This first rule is not needed for completeness. However it may improve efficiency and readability of the deduction process.

The second rule, called **Propagate** allows propagating the value of a solved variable to the first-order part. It is needed for the completeness of the deduction process, as illustrated by the next example.

Example 6 *Consider the two constrained rules:*

$$x \rightarrow a,\ (x * x * x * x =_{AC}^? u * v * w * a)$$
$$x' * x' * x' * x' \rightarrow x',\ \top.$$

where $*$ is an associative commutative symbol. A possible instantiation of the first rule is given by the substitution α defined as $\{(x \mapsto a * b * b * b * z)\}$ and the subterm $b * b * b * z$ is unifiable with $x' * x' * x' * x'$ using $\{(x' \mapsto b)(z \mapsto b)\}$. This superposition yields a critical pair $\langle \sigma(x), a \rangle$ where $\sigma(x) = a * b$. However this critical pair is not computed by the constrained deduction rule **Deduce**. *It is then needed to solve the constraint $(x * x * x * x =^{?}_{AC} u * v * w * a)$ and to propagate the solutions into the rules.*

Another method will be proposed in Section 9 to deal with this completeness problem.

Elim Or	$CF \cup (F, c_1 \vee c_2)$	$\mapsto\!\!\!\!\rightarrow$	$CF \cup \{(F, c_1), (F, c_2)\}$
Propagate	$CF \cup (F, c \wedge (x =^{?}_{\emptyset} t))$	$\mapsto\!\!\!\!\rightarrow$	$CF \cup (\sigma(F), c)$ if $x \notin \mathcal{V}(t) \cup \mathcal{V}(c), \sigma = (x \mapsto t)$

Restructuring Rules

Proposition 7 *The term-generated models of a set of constrained formulae are preserved when* **Elim Or** *or* **Propagate** *are applied.*

Proof: This is a consequence of the following points:

- if σ is solution of $c_1 \vee c_2$, then σ is solution of either c_1 or c_2.
- if σ is solution of $c_1 \wedge c_2$, then σ is solution of both c_1 and c_2.

□

7 Bounded deduction for first-order logic with equality

In this section, by setting more control on the inference rules defined in Section 5 and 6, we derive a powerful theorem proving strategy. This control is a natural one: it prescribes to solve at first the equational constraints and propagate solutions immediately, using the restructuring rule **Propagate**. Hence, the only constraints whose resolution may be delayed are the inequational ones.

The strategy is refutationally complete by a straightforward modification of results in [Rus88], and by a lifting lemma for the constrained paramodulation and the constrained superposition rules. The strategy is indeed more powerful than the *superposition strategy* of [Rus88] since it may detect the satisfiability of clause sets upon which the *ordered strategy* loops.

7.1 Inference rules

As usual, we assume that $\prec_{\emptyset}$ is a complete simplification ordering. We give now three inference rules for ground clauses. If we assume that the clause sets contain

all the identities $a = a$ for any ground term a, then the rules are complete for refutation as it is proved in [Rus88]:

GOFactoring
$CF \cup \{L_0 \vee L_1 \vee \cdots \vee L_n\}$
$\mapsto\!\!\!\!\rightarrow$
$CF \cup \{L_0 \vee L_1 \vee \cdots \vee L_n\} \cup \{L_1 \vee L_2 \vee \cdots \vee L_n\}$
if $L_0 = L_1$, $\forall i, i \leq n, L_i \preceq_\emptyset L_0$

GOResolution
$CF \cup \{L_0 \vee L_1 \vee \cdots \vee L_n, \neg L'_0 \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m\}$
$\mapsto\!\!\!\!\rightarrow$
$CF \cup \{L_0 \vee L_1 \vee \cdots \vee L_n, \neg L'_0 \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m\}$
$\cup \{L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m\}$
if $L_0 = L'_0$, $\forall 0 < i \leq n, L_i \prec_\emptyset L_0$,
$\forall 0 < i \leq m, L'_i \prec_\emptyset L'_0$

GOParamodulation
$CF \cup \{s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, L'[s] \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m\}$
$\mapsto\!\!\!\!\rightarrow$
$CF \cup \{s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, L'[u] \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m\}$
$\cup \{L'[t] \vee L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m\}$
if $t \prec_\emptyset s$, $\forall 0 < i \leq m, L'_i \prec_\emptyset L'[s]$

GOSuperposition
$CF \cup \{s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, a[s] = b \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m\}$
$\mapsto\!\!\!\!\rightarrow$
$CF \cup \{s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, a[s] = b \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m\}$
$\cup \{a[t] = b \vee L_1 \vee L_2 \vee \cdots \vee L_n \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m$
if $t \prec_\emptyset s$, $b \prec_\emptyset a[s]$, $\forall 0 < i \leq m$, $L'_i \prec_\emptyset a[s] = b$

Rules for ground clauses

These inference rules have been lifted to clauses with variables in [HR86, Rus88] by introducing unification and by replacing any symbol $\prec_\emptyset$ by $\not\preceq_\emptyset$, and $\preceq_\emptyset$ by $\not\prec_\emptyset$. This process is rather brutal, so we introduce another way to lift the rules, which is to make explicit the ground instantiations that matter. We get the following rules:

BOF

$$CF \cup \{(L_0 \vee L_1 \vee \cdots \vee L_n, c)\}$$
$$\mapsto\!\!\!\!\rightarrow$$
$$CF \cup \{(L_0 \vee L_1 \vee \cdots \vee L_n, c)\} \cup \{(\sigma(L_1) \vee \cdots \vee \sigma(L_n), \sigma(c'))\}$$
if $\sigma = mgu(L_0, L_1)$ *and* $c' = c \wedge (\bigwedge_{i=2}^{n} L_i \preceq_{\emptyset}^{?} L_0)$

BOR

$$CF \cup \{(L_0 \vee L_1 \vee \cdots \vee L_n, c), (\neg L'_0 \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$$
$$\mapsto\!\!\!\!\rightarrow$$
$$CF \cup \{(L_0 \vee L_1 \vee \cdots \vee L_n, c), (\neg L'_0 \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$$
$$\cup\{(\sigma(L_1) \vee \cdots \vee \sigma(L_n) \vee \sigma(L'_1) \vee \cdots \vee \sigma(L'_m), \sigma(c''))\}$$
if $\sigma = mgu(L_0, L'_0)$ *and*
$c'' = c \wedge c' \wedge (\bigwedge_{i=1}^{n} L_i \prec_{\emptyset}^{?} L_0) \wedge (\bigwedge_{i=1}^{m} L'_i \prec_{\emptyset}^{?} L'_0)$

BOP

$$CF \cup \{(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c), (L'[u] \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$$
$$\mapsto\!\!\!\!\rightarrow$$
$$CF \cup \{(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c), (L'[u] \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$$
$$\cup\{(\sigma(L'[t]) \vee \sigma(L_1) \vee \cdots \vee \sigma(L_n) \vee \sigma(L'_1) \vee \cdots \vee \sigma(L'_m), \sigma(c''))\}$$
if $\sigma = mgu(s, u), u \notin X$ *and*
$c'' = c \wedge c' \wedge (\bigwedge_{i=1}^{m} L'_i \prec_{\emptyset}^{?} L'[u]) \wedge (t \prec_{\emptyset}^{?} s)$

BOS

$$CF \cup \{(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c), (a[u] = b \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$$
$$\mapsto\!\!\!\!\rightarrow$$
$$CF \cup \{(s = t \vee L_1 \vee L_2 \vee \cdots \vee L_n, c), (a[u] = b \vee L'_1 \vee L'_2 \vee \cdots \vee L'_m, c')\}$$
$$\cup\{(\sigma(a[t] = b) \vee \sigma(L_1) \vee \cdots \vee \sigma(L_n) \vee \sigma(L'_1) \vee \cdots \vee \sigma(L'_m), \sigma(c''))\}$$
if $\sigma = mgu(s, u), u \notin X$ *and*
$c'' = c \wedge c' \wedge (\bigwedge_{i=1}^{m} L'_i \prec_{\emptyset}^{?} a[u] = b) \wedge (t \prec_{\emptyset}^{?} s) \wedge (b \prec_{\emptyset}^{?} a[u])$

Bounded inference rules for first-order logic

The correctness of these rules is just a special case of the correctness of the constrained inference rules of Section 5.

7.2 Refutation completeness

Refutation completeness of the **BOF**, **BOR**, **BOP** and **BOS** rules is obtained by a straightforward adaptation of the *semantic tree technique* of [Rus89]. If a set of clauses is unsatisfiable (i.e. has no model), every branch of its semantic tree has a failure node which can be labelled by a ground instance of a clause. If every possible inference has been applied to the clauses, then given some failure node, it is always possible to discover another one which is closer to the root of the tree: this implies that the root itself is a failure node; hence, the empty clause has been generated. The needed inference steps on ground clauses must generate clauses which are also instances of clauses which are deduced at the first-order level. This is ensured by the so-called *lifting lemmas*. Such lemmas exist for each of our inference rules. We just present here a paramodulation lifting

lemma which takes into account the chosen ordering. For simplicity, we ignore the factoring steps needed. The lifting arguments are very similar to the case of resolution lifting given in [Pet83].

Lemma 1 (BOP-rule Lifting Lemma) *Let (C_1, c_1) and (C_2, c_2) be two variable disjoint constrained clauses and let n be a non-variable position in C_2. Let D be an O-paramodulant from $\sigma(C_1)$ into $\sigma(C_2)$ at n, where σ is a ground substitution, such that $\sigma(c_1 \wedge c_2)$ is satisfiable in $T(F)$, then there is an OP-paramodulant (C, c) from (C_1, c_1) into (C_2, c_2) at n such that D is an instance of C.*

Proof: Let C_1 be $(s = t) \vee C_1'$, C_2 be $L \vee C_2'$, and θ be a ground substitution such that $\theta(L) \succ \theta(C_2')$, $\theta(s) \succ \theta(t)$, and $\theta(s = t) \succ \theta(C_1')$. Furthermore, suppose n is a non-variable position in the literal L and let D be the O-paramodulant $\theta(C_2)[\theta t]_n \vee \theta(C_1')$. Since n is a position in C_2, $\theta(C_2)_{|n} = \theta(C_{2|n})$. Thus, $\theta(C_{2|n}) = \theta(s)$ since $\theta(s) = \theta(C_2)_{|n}$. Therefore $C_{2|n}$ and s are unifiable. Let σ be their most general unifier. Then there is a substitution ψ such that $\theta = \psi\sigma$. Moreover, the constraint $\sigma(t) \prec_\emptyset^? \sigma(s) \wedge \sigma(C_2') \prec_\emptyset^? \sigma(L) \wedge \sigma(C_1') \prec_\emptyset^? \sigma(s = t)$ is satisfiable since it has ψ for solution. Therefore there is an OP-paramodulant (C, c) from (C_1, c_1) into (C_2, c_2) at n such that c is $\sigma(t) \prec_\emptyset^? \sigma(s) \wedge \sigma(C_2') \prec_\emptyset^? \sigma(L) \wedge \sigma(C_1') \prec_\emptyset^? \sigma(s = t)$ and $C = \sigma(C_2[t]_n \vee C_1')$, and $\psi(C) = \psi\sigma(C_2[t]_n \vee C_1') = \theta(C_2[t]_n \vee C_1') = \theta(C_2)[\theta t]_n \vee \theta(C_1') = D$. □

Before stating the completeness theorem, let us recall the definition of *fairness*. Let $S_0, S_1, S_2, \cdots$ be a sequence of sets of clauses, such that every set S_{i+1} is obtained from S_i by an inference (**BOF**, **BOR**, or **BOP**). Then the sequence is *fair* if every clause which can be derived in one step from $\bigcup_i \bigcap_{j \geq i} S_j$ by the rules above is contained in $\bigcup_i S_i$.

Theorem 1 *If $S_0, S_1, S_2, \cdots$ is a fair sequence where S_0 is unsatisfiable and contains the clause $x = x$, then some set S_k contains the empty clause.*

7.3 Subsumption and Simplification

We can also derive constrained versions for subsumption and simplification (also called demodulation in the context of first-order theorem proving). They are very important with regard to efficiency, since subsumption is used to get rid of redundancy, and simplification is used to keep clauses in a normalized format. Constrained simplification has been introduced in Section 4.2. Let us give now a constrained version of the subsumption rule:

Definition 20 *The constrained clause (C, c) subsumes the constrained clause (C', c') if there is a substitution θ such that:*

1. *c' implies $\theta(c)$ (in $T(F)$).*

2. *The number of literals of C is smaller than or equal to the number of literals of C'.*

3. *$\theta(C)$ is a sub-clause of C'.*

Condition 2 prevents from subsuming a clause by one of its factors. Completeness is retained since the subsuming clause schematized a larger set of ground clauses than the subsumed one.

8 Unfailing bounded completion

The Knuth-Bendix *completion* procedure [KB70] is aimed at building a Church-Rosser and terminating rewrite system from a set of equalities. Completion computes critical pairs, orients equalities into rewrite rules and keeps terms in normal form for the current set of rewrite rules. Completion procedures can abort on an equality that cannot be oriented into a terminating rule. In the *vunfailing completion* [BDP89, HR87], equalities are always used to deduce equational consequences (i.e. for the computation of critical pairs) but their use for reduction is limited. When it terminates, the unfailing completion returns a set of rewrite rules and a set of equalities that provide a decision method for ground theorems in the original theory [BDP89, HR87].

By considering constrained rules, we show now that it is possible to deal with non-orientable rules, in a more general way than in [BDP89, HR87]. In particular, the divergence problem of completion can be better controlled: constrained completion often yields ground Church-Rosser systems where unfailing completion diverges.

The correctness of this procedure follows from the refutational completeness of the inference rules of Section 7.1. Indeed, the procedure is obtained by specializing these rules to equations [Rus89].

We shall restrict here to completion in the empty theory. Moreover, the equational problems will be solved immediately and only inequational constraints will be left unsolved, applying the same strategy as in bounded clausal theorem proving.

We shall use essentially total simplification, for sake of termination. The general simplification rule, which generates two equations from one, will be applied only in connection with the **Delete** rule, in the so-called joinability test defined below.

Definition 21 *A constrained equation $(s = t, c)$ is* joinable *by R if one of the following cases occurs:*

1. *s and t are identical*

2. *c is equivalent to $\perp$ (false)*

3. *$(s = t, c)$ can be simplified by R to a set of equations, each of which being joinable.*

When a constrained equation is joinable, each of the schematized ground equations can be normalized to a trivial one. But the way to perform this normalization may depend of the instance. That is why this technique to detect trivial critical pairs is more powerful than the total simplification one which is used in classical completion procedures.

For **Simplify** and **Collapse**, we shall check that the rule used for simplification is smaller than the equation (or the rule) it simplifies w.r.t. a well-founded ordering $\lhd$ which extends $\prec_\emptyset$. For instance, we can define $\lhd$ as the transitive closure of the relation (also denoted by $\lhd$) defined by: $(l = r, c') \lhd (p = q, c)$ if there exists an occurrence m of $p = q$ and a substitution θ such that $(p = q)_{|m} = \theta(l)$ and such that for any solution σ of $\theta(c) \wedge \theta(c')$, we have $\sigma(l = r) \prec_\emptyset \sigma(p = q)$. Less restrictive conditions for applying simplification can be found in [Bac87].

Deduce	CP, CR
	$\mapsto\!\!\!\!\rightarrow$
	$CP \cup \{\sigma(l[t] = r, c \wedge c')\}, CR$
	if $(l[u] \rightarrow r, c), (s \rightarrow t, c') \in R, \sigma = mgu(s, u)$
Delete	$CP \cup \{(p = q, c)\}, CR$
	$\mapsto\!\!\!\!\rightarrow$
	CP, CR
	if $(p = q, c)$ *is joinable*
Orient	$CP \cup \{(p = q, c)\}, CR$
	$\mapsto\!\!\!\!\rightarrow$
	$CP, CR \cup \{(p \rightarrow q, (c \wedge (q \prec^?_\emptyset p))),$
	$(q \rightarrow p, (c \wedge (p \prec^?_\emptyset q)))\}$
Simplify	$CP \cup \{(p = q, c)\}, CR$
	$\mapsto\!\!\!\!\rightarrow$
	$CP \cup \{(p' = q, c')\}, CR$
	if $(p = q, c) \rightarrow\!\!\!\!\!\rightarrow (p' = q, c)$ *using* $(g \rightarrow d, c'')$
	s.t. $(g = d, c'') \lhd (p = q, c)$
Compose	$CP, CR \cup \{(l \rightarrow r, c)\}$
	$\mapsto\!\!\!\!\rightarrow$
	$CP, CR \cup \{(l \rightarrow r', c')\}$
	if $(l \rightarrow r, c) \rightarrow\!\!\!\!\!\rightarrow (l \rightarrow r', c')$
Collapse	$CP, CR \cup \{(l \rightarrow r, c)\}$
	$\mapsto\!\!\!\!\rightarrow$
	$CP \cup \{(l' = r, c)\}, CR\}$
	if $(l = r, c) \rightarrow\!\!\!\!\!\rightarrow (l' = r, c)$ *using* $(g \rightarrow d, c'')$
	s.t. $(g = d, c'') \lhd (l = r, c)$

Rules for bounded completion

The next example illustrates the bounded completion procedure.

Example 7 *We consider here an initial system with two equations on the sig-*

nature $F = \{+, 0\}$ *and the precedence* $0 \prec +$*:*

$$\begin{aligned} x + 0 &= 0 + x \\ (x + y) + z &= x + (y + z) \end{aligned}$$

which are converted into constrained rules:

$$(x + 0 \to 0 + x,\ 0 \prec^?_\emptyset x) \tag{1}$$
$$((x + y) + z \to x + (y + z),\ \top) \tag{2}$$

Two constrained superpositions can be computed from these rules:

$$((0 + x) + z = x + (0 + z),\ 0 \prec^?_\emptyset x)$$
$$(0 + (x + y) = x + (y + 0),\ \ 0 \prec^?_\emptyset (x + y))$$

Let us simplify the constraints. The first equality can be oriented. We get:

$$(x + (0 + z) \to 0 + (x + z),\ 0 \prec^?_\emptyset x) \tag{3}$$
$$(0 + (x + y) = x + (y + 0),\ \top) \tag{4}$$

Let us prove that (4) is joinable. (4) is simplified by (1) into the following ones:

$$(0 + (x + y) = x + (0 + y),\ 0 \prec^?_\emptyset y) \tag{5}$$
$$(0 + (x + y) = x + (y + 0),\ y \prec^?_\emptyset 0 \vee y =^?_\emptyset 0) \tag{6}$$

The equation (5) can be simplified by (3) and yields:

$$(0 + (x + y) = 0 + (x + y),\ 0 \prec^?_\emptyset y \wedge 0 \prec^?_\emptyset x) \tag{7}$$
$$(0 + (x + y) = x + (0 + y),\ 0 \prec^?_\emptyset y \wedge (x \prec^?_\emptyset 0 \vee x =^?_\emptyset 0)) \tag{8}$$

These equations are easily proved to be joinable. The equation (6) is first transformed into

$$(0 + (x + 0) = x + (0 + 0),\ \top) \tag{9}$$

Then (9) is joinable since it can be also simplified by (1) into:

$$(0 + (0 + x) = x + (0 + 0),\ 0 \prec^?_\emptyset x) \tag{10}$$
$$(0 + (x + 0) = x + (0 + 0),\ (x \prec^?_\emptyset 0 \vee x =^?_\emptyset 0)) \tag{11}$$

The equation (11) is trivial and deleted. (10) can be simplified by (3) to give also a trivial equation. The superpositions of (2) and (3) are:

$$(0 + ((x + z) + z') = x + ((0 + z) + z'),\ 0 \prec^?_\emptyset x)$$
$$(0 + ((x + y) + z') = x + ((y + 0) + z'),\ 0 \prec^?_\emptyset x + y)$$

Both are joinable. The remaining equations are (1), (2), (3) which constitute a canonical system:

$$(x + 0 \to 0 + x,\ 0 \prec^?_\emptyset x)$$
$$((x + y) + z \to x + (y + z),\ \top)$$
$$(x + (0 + z) \to 0 + (x + z),\ 0 \prec^?_\emptyset x)$$

This example can be generalized to the commutativity and associativity axioms:

$$\begin{aligned} x + y &= y + x \\ (x + y) + z &= x + (y + z) \end{aligned}$$

Following U. Martin and T. Nipkow [MN90] or G. Peterson [Pet90], we can obtain too:

$$(x + y \to y + x,\ y \prec^?_\emptyset x)$$
$$((x + y) + z \to x + (y + z),\ \top)$$
$$(x + (y + z) \to y + (x + z),\ y \prec^?_\emptyset x)$$

9 Application to completion modulo T

We consider in this section the problem of completion modulo a set of axioms T [BD89, JK86]. A main drawback of this class of completion procedures is their inefficiency, due to the computation of matches and unifiers modulo T. Here constraints are used to record unification problems in the theory T and to avoid solving them immediately.

We use the following notations: given a set T of equalities, a set R of constrained rewrite rules, and the set of schematized rules $\mathcal{S}(R)$, $\longleftrightarrow^T$ denotes one step of replacement of equals by equals in T, $\overset{*}{\longleftrightarrow}^T$ or $=_T$ denotes the reflexive symmetric transitive closure of the previous relation, and $\to^{\mathcal{S}(R),T}$ denotes equational rewriting modulo T with rules in $\mathcal{S}(R)$ [PS81]. The class rewriting relation modulo T is denoted by $\to^{\mathcal{S}(R)/T}$. Taking $T = \emptyset$ gives the corresponding results for standard rewriting.

We apply in this section the previous deduction rules, but we make more precise the relationships between constrained superposition and rewriting on one hand, and superposition and rewriting with the schematized rules and equalities, on the other hand. This may be understood as the operational semantics of these constrained deduction rules.

9.1 Specific T-constraints

Concerning equations modulo T, complete sets of solutions are usually considered in the computation of critical pairs. We have defined, in Definition 7 of

Section 2.3, a similar notion for constraints that will be extensively used in this section. Actually, we can only consider complete sets of solutions for constraints, without changing the semantics of equality. More precisely,

Lemma 2 *Let E be a set of constrained equalities, $(l = r, c) \in E$, and*

$$CSS(l = r, c) = \{\sigma(l) = \sigma(r) | \alpha \in CSS(c),\ \sigma = \alpha_{|\mathcal{V}(c)}\}.$$

Then $=_{\mathcal{S}(E)\cup T}$ and $=_{CSS(E)\cup T}$ coincide on $\mathcal{T}(F)$.

The same lemma holds of course for rewrite rules.

In the last result, restricting the domain of substitutions used in the instantiation to the constrained variables allows focussing on useful instances of the constrained equality.

Concerning elementary constraints that are not equations, we need the additional assumption that any inequation $(t \prec_T^? t')$ and any disequation $(t \neq_T^? t')$ has a solved form which a union of systems $(\bigwedge_i (x_i =_\emptyset^? t_i)$. As a consequence, a solved form system may be identified with a substitution. This hypothesis is used in the proofs of the completion process given in this section and usually implies some restrictions to the chosen model of interest.

We actually need more about inequations modulo T. We gave in Section 4.4 a general form for the orientation rule. Here we assume that $\prec_T$ is a reduction ordering compatible with T and total on equivalence classes modulo T. This allows avoiding negation for inequations. So we get an unfailing completion modulo T, but as a counterpart, the Church-Rosser property only holds on ground terms.

An alternative to this hypothesis of a total ordering, proposed in [KK89], is to develop the non-orientable instances of $(p = q, c)$; the following set of unconstrained pairs is thus computed:

$$\{(\sigma(p) = \sigma(q), \top) | \sigma \in CSS(c \wedge (\neg(p \prec_T^? q) \wedge \neg(p \prec_T^? q)))\}.$$

This method assumes again that any inequation $(t \prec_T^? t')$ has a solved form which a union of systems $(\bigwedge_i (x_i =_\emptyset^? t_i)$. Note also that before developing, the satisfiability check is crucial.

9.2 Constrained superposition modulo T

The operational semantics of constrained superposition is refined by the following result, in the case of rewrite rules.

Proposition 8 *Let $(\langle g[r]_m, d\rangle, c \wedge c' \wedge (g_{|m} =_T^? l))$ be the constrained critical pair obtained by superposition of $(l \to r, c)$ into $(g \to d, c')$ at a non-variable occurrence m. Let $W = \mathcal{V}(l) \cup \mathcal{V}(r) \cup \mathcal{V}(c)$ and $W' = \mathcal{V}(g) \cup \mathcal{V}(d) \cup \mathcal{V}(c')$ and assume that $W \cap W' = \emptyset$.*

Then for any solution $\sigma \in Sol(c \wedge c' \wedge (g_{|m} =_T^? l))$, there exist $\alpha \in CSS(c)$, $\beta \in CSS(c')$ and $\sigma' \leq_T^{W \cup W'} \sigma$, such that $\langle \sigma'(g[r]_m), \sigma'(d)\rangle$ is a critical pair of $(\alpha(l) \to \alpha(r))$ into $(\beta(g) \to \beta(d))$ at occurrence m.

Proof: Using the fact that σ is solution of c and c', we get $\alpha, \mu_1, \beta, \mu_2$ such that $\sigma =_T^W \mu_1 \circ \alpha$ and $\sigma =_T^{W'} \mu_2 \circ \beta$. Since $\sigma(g_{|m}) =_T \sigma(l)$, $\mu_1 \circ \mu_2$ T-unifies $\alpha(l)$ and $\beta(g)_{|m}$. So there exist μ in a complete set of T-unifiers of these two terms, and some substitution γ such that $\sigma =_T^{W \cup W'} \gamma \circ \mu \circ \alpha \circ \beta$. Note that $\mu \circ \alpha \circ \beta$ is a solution of $(c \wedge c' \wedge (g_{|m} =_T^? l))$. So there is a critical pair $\langle \mu(\beta(g)[\alpha(r)]_m), \mu(\beta(d)) \rangle$ obtained by superposition of $(\alpha(l) \rightarrow \alpha(r))$ into $(\beta(g) \rightarrow \beta(d))$ at occurrence m. This critical pair can be written as $\langle \sigma'(g[r]_m), \sigma'(d) \rangle$, with $\sigma' = \mu \circ \alpha \circ \beta$, and $\sigma' \leq_T^{W \cup W'} \sigma$. □

Since we would like to deal with completion modulo a theory T, the superposition of a constrained rule into an axiom of T has to be considered too. We can apply the superposition rule to the case where (F, c) is an unconstrained equality in T, and get a similar result to Proposition 8 about critical pairs between rules and axioms. But we can also obtain the concept of *constrained extension* that schematizes the extension rules used in completion modulo T.

Definition 22 *The constrained rewrite rule $(l \rightarrow r, c)$ overlaps the axiom $(g = d) \in T$ at the non-variable occurrence m in g if the constraint $(c \wedge (g_{|m} =_T^? l))$ is satisfiable. The* constrained extended rule *obtained by superposition of $(l \rightarrow r, c)$ into $(g = d)$ at occurrence m is by definition:*

$$(g[l]_m \rightarrow g[r]_m, c \wedge (g_{|m} =_T^? l)).$$

The operational semantics of constrained extended rules is given in a similar way:

Proposition 9 *Let $(g[l]_m, g[r]_m, c'')$ be the constrained extended rule obtained by superposition of $(l \rightarrow r, c)$ into $(g = d)$ at occurrence m. Let $W = \mathcal{V}(l) \cup \mathcal{V}(r) \cup \mathcal{V}(c)$ and $W' = \mathcal{V}(g) \cup \mathcal{V}(d)$ and assume that $W \cap W' = \emptyset$.*

Then for any solution $\sigma \in Sol(c \wedge (g_{|m} =_T^? l))$, there exist $\alpha \in CSS(c)$, $\alpha \leq_T^{W \cup W'} \sigma$, such that $(\alpha(g[l]_m) \rightarrow \alpha(g[r]_m))$ is an extended rule for $(\alpha(l) \rightarrow \alpha(r))$.

Proof: Using the fact that σ is solution of c, we get α, μ_1, μ_2 such that $\sigma =_T^W \mu_1 \circ \alpha$ and $\sigma =_\emptyset^{W'} \mu_2$. Since $\sigma(g_{|m}) =_T \sigma(l)$, $\mu_1 \circ \mu_2$ T-unifies $\alpha(l)$ and $g_{|m}$. So there exist μ in a complete set of T-unifiers of these two terms, and some substitution γ such that $\sigma =_T^{W \cup W'} \gamma \circ \mu \circ \alpha$. Note that $\mu \circ \alpha$ is a solution of $(c \wedge (g_{|m} =_T^? l))$. So there is an extended rule for $(\alpha(l) \rightarrow \alpha(r))$, namely $(g[\alpha(l)]_m \rightarrow g[\alpha(r)]_m)$. It can also be written as $(\alpha(g[l]_m) \rightarrow \alpha(g[r]_m))$. □

Example 8 *In the considered AC-theory, the constrained rule*

$$(x * x * x * x \rightarrow x, \ \bot)$$

needs an extension

$$(x * x * x * x * z' \rightarrow x * z', \ x * x * x * x =_{AC}^? x' * y')\}).$$

with respect to the associativity axiom $(x' * y') * z' = x' * (y' * z')$.

9.3 Constrained simplification modulo T

The next result states an operational relation between constrained simplification and usual simplification of schematized formulae.

Proposition 10 *If* $(F_1, c_1) \rightarrow\!\!\!\!\rightarrow^T \{(F_2, c_2), (F_1, c_2')\}$ *with the constrained rewrite rule* $(l \rightarrow r, c)$,

- *for any solution* σ *of* c_2, *there exists a solution* α *in* $CSS(c)$ *such that*

$$\sigma(F_1) \rightarrow^{\alpha(g)\rightarrow\alpha(d),T} F_2' =_T \sigma(F_2).$$

- *If* c_2' *is satisfiable, for any solution* σ' *of* c_2', $\sigma'(F_1)$ *is irreducible, at position* m, *by all instances* $(\sigma(l) \rightarrow \sigma(r))$ *such that* $\sigma \in Sol(c)$.

Proof:

- Let W be $\mathcal{V}(l) \cup \mathcal{V}(r) \cup \mathcal{V}(c)$. Since σ is solution of c_2, σ is also solution of c and there exists a solution α in $CSS(c)$ such that $\sigma \leq_T^W \beta \circ \alpha$, for some substitution β.
  Since $\beta(\alpha(l)) =_T \sigma(F_{1|m})$,

$$\sigma(F_1) \rightarrow^{\alpha(l)\rightarrow\alpha(r),T} \sigma(F_1)[\beta(\alpha(r))]_m = F_2' =_T \sigma(F_1[r]_m) = \sigma(F_2).$$

- Assume that c_2' is satisfiable; for any solution σ' of c_2', if $\sigma'(F_1)$ is reducible, at position m, by an instance $(\sigma(l) \rightarrow \sigma(r))$ such that $\sigma \in Sol(c)$, this would imply that $c_1 \wedge c \wedge (F_{1|m} =_T^? l)$ is satisfiable. That is the formula $\exists W. c_1 \wedge c \wedge (F_{1|m} =_T^? l)$, with $W = \mathcal{V}(l) \cup \mathcal{V}(r) \cup \mathcal{V}(c)$, is valid. This contradicts the fact that the negation of this formula, namely c_2', is satifiable by σ'.

□

9.4 Superposition in constraints

As shown in Example 6, the inference rule **Deduce** is not complete for the computation of critical pairs. In order to get back a completeness result, a first possibility is to solve constraints, to propagate the solutions into the rules and to continue the completion process. We propose here another method whose advantage is to stay in a constraint formalism, while progressively solving constraints. This method consists of considering superpositions of a constrained rewrite rule $(l \rightarrow r, c)$ into an equation of the form $(x =_\emptyset^? t)$ in the constraint c' of the constrained rule $(g \rightarrow d, c')$. If such an equation does not exist in c', then the solving process on the constraint c' must be applied, until getting it. This solving process must terminate, since we assume the existence of a canonical solved form $c'\downarrow$ for the constraint c'.

The superposition in constraints can be applied as soon as a solved form relative to a chosen variable x is reached.

Definition 23 *A symbolic constraint c' is* in solved form with respect to a variable x *if* $x \in \mathcal{V}(c')$, $(x=^?_\emptyset t)$ *belongs to* c', $x \notin \mathcal{V}(t)$ *and* x *does not occur anywhere else in* c'.

Then this solved form for c' with respect to a variable x in the left-hand side of a constrained rule $(g \to d, c')$ gives a partial instantiation that produces a critical pair:

Definition 24 *Let $(g \to d, c')$ be a constrained rewrite rule such that c' is in solved form with respect to the variable x and $x \in \mathcal{V}(g)$. The constrained rewrite rule $(l \to r, c)$ overlaps in constraints $(g \to d, c')$ at the non-variable occurrence m in $(x=^?_\emptyset t) \in c'$ if $(c \wedge c' \wedge (t_{|m}=^?_T l))$ is satisfiable.*

The critical pair in constraints *obtained by superposition of $(l \to r, c)$ into $(g \to d, c')$ at occurrence m in $(x=^?_\emptyset t) \in c'$ is by definition:*

$$(\langle g, d\rangle, c \wedge (c' - (x=^?_\emptyset t)) \wedge (x=^?_\emptyset t[r]_m) \wedge (t_{|m}=^?_T l)).$$

The operational semantics of superposition in constraints is given in a precise way by the following result:

Proposition 11 *Let $(\langle g, d\rangle, c \wedge (c' - (x=^?_\emptyset t)) \wedge (x=^?_\emptyset t[r]_m) \wedge (t_{|m}=^?_T l))$ be the critical pair in constraints obtained by superposition of $(l \to r, c)$ into $(g \to d, c')$ at occurrence m in $(x=^?_\emptyset t) \in c'$, where c' is in solved form with respect to x. Let $W = \mathcal{V}(l) \cup \mathcal{V}(r) \cup \mathcal{V}(c)$ and $W' = \mathcal{V}(g) \cup \mathcal{V}(d) \cup \mathcal{V}(c')$ and assume that $W \cap W' = \emptyset$. Let n be an occurrence of x in g.*

Then for any solution $\sigma \in Sol(c \wedge (c' - (x=^?_\emptyset t)) \wedge (x=^?_\emptyset t[r]_m) \wedge (t_{|m}=^?_T l))$, there exist $\alpha' \in CSS(c)$, $\beta' \in CSS(c')$ and a critical pair $\langle p, q\rangle$ of $(\alpha'(l) \to \alpha'(r))$ into $(\beta'(g) \to \beta'(d))$ at occurrence $n.m$, such that

$$\gamma(p) \xrightarrow{*}^{\mathcal{S}(R)/T} \sigma(g) \text{ and } \gamma(q) \xrightarrow{*}^{\mathcal{S}(R)/T} \sigma(d)$$

for some substitution γ.

Proof: From σ, let deduce α such that $\alpha(y) = \sigma(y)$ for $y \neq x$ and $\alpha(x) = t$. Let n be one of the (possibly many) occurrences of the variable x in g. Since α is solution of $(c \wedge c' \wedge (t_{|m}=^?_T l))$, $\alpha(t_{|m}) =_T \alpha(l)$, $\alpha(g)_{|n.m} =_T \alpha(l)$. Then we get $\alpha', \mu_1, \beta', \mu_2$ such that $\alpha =^W_T \mu_1 \circ \alpha'$ and $\alpha =^{W'}_T \mu_2 \circ \beta'$. Since $\alpha(g)_{|n.m} =_T \alpha(l)$, $\mu_1 \circ \mu_2$ T-unifies $\alpha'(l)$ and $\beta'(g)_{|n.m}$. So there exist μ in a complete set of T-unifiers of these two terms, and some substitution γ such that $\alpha =^{W \cup W'}_T \gamma \circ \mu \circ \alpha' \circ \beta'$. So there is a critical pair $\langle p, q\rangle = \langle \mu(\beta'(g)[\alpha'(r)]_{n.m}), \mu(\beta'(d))\rangle$ obtained by superposition of $(\alpha'(l) \to \alpha'(r))$ into $(\beta'(g) \to \beta'(d))$ at occurrence $n.m$. This critical pair can be written as $\langle \sigma'(g)[\sigma'(r)]_{n.m}, \sigma'(d)\rangle$, with $\sigma' = \mu \circ \alpha' \circ \beta'$, and $\sigma' \leq^{W \cup W'}_T \alpha$. Moreover $\gamma(p) =_T \alpha(g) \xrightarrow{*}^{\alpha'(l) \to \alpha'(r), T} \sigma(g)$ and $\gamma(q) =_T \alpha(d) \xrightarrow{*}^{\alpha'(l) \to \alpha'(r), T} \sigma(d)$. □

9.5 A critical pair lemma

We now prove an important lemma that shows how to transform special kinds of proofs called peaks and cliffs [Bac87]. This result also justifies the use of complete sets of solutions for constraints during the completion process.

Lemma 3 *Consider two constrained rules* $(g \to d, c')$ *and* $(l \to r, c)$ *and any peak*

$$t'' \longleftarrow^{\alpha(l)\to\alpha(r),T} t \longrightarrow^{\beta(g)\to\beta(d),T} t'$$

such that $\alpha \in CSS(c)$ *and* $\beta \in CSS(c')$. *Then*

- *either* $t'' \longrightarrow^{*}_{\mathcal{S}(R),T} =_T \longleftarrow^{*}_{\mathcal{S}(R),T} t'$,
- *or there exists a constrained critical pair or a critical pair in constraints of the constrained rewrite rules* $(l \to r, c)$ *into* $(g \to d, c')$, *say* $(\langle p, q\rangle, c_0)$, *such that there are substitutions* $\alpha \in CSS(c_0)$ *and* β *satisfying* $t'' \longrightarrow^{*}_{\mathcal{S}(R),T} =_T \beta(\alpha(p))$ *and* $t' \longrightarrow^{*}_{\mathcal{S}(R),T} =_T \beta(\alpha(q))$.

Consider a constrained rule $(l \to r, c)$ *and an axiom* $(g = d)$ *in* T, *and any cliff*

$$t'' \longleftarrow^{\sigma(l)\to\sigma(r),T} t \longleftrightarrow^{g=d} t'$$

such that $\sigma \in CSS(c)$. *Then*

- *either* $t'' \longrightarrow^{*}_{\mathcal{S}(R),T} =_T \longleftarrow^{*}_{\mathcal{S}(R),T} t'$,
- *or there exists a constrained extended rule* $(g[l]_m \to g[r]_m, c_1)$ *of the constrained rewrite rule* $(l \to r, c)$ *with respect to* $(g = d)$, *such that there are substitutions* $\alpha \in CSS(c_1)$ *and* β *satisfying* $t' =_T \beta(\alpha(g[l]_m))$ *and* $t'' =_T \beta(\alpha(g[r]_m))$.

Proof: Consider first the case of a peak. If the rewriting steps apply on disjoint subterms of t, then they simply commute. The other case to be considered is when one rewriting applies above the other. This case can in turn be reduced to the case of a peak

$$t'' \longleftarrow^{\sigma(l)\to\sigma(r),T} t \longrightarrow^{\sigma(g)\to\sigma(d)} t'$$

such that $\sigma \in CSS(c \wedge c')$ and where the $\longleftarrow$-rewrite step modulo T occurs below the $\longrightarrow$-rewrite step.

Without lost of generality, we can assume that $(\sigma(g) \to \sigma(d))$ applies at position ϵ in t, thus $t' = \mu(\sigma(d))$, and $(\sigma(l) \to \sigma(r))$ applies at some occurrence m below. It can also be assumed that $\mathcal{V}(t)$ does not intersect either $W = \mathcal{V}(l) \cup \mathcal{V}(r) \cup \mathcal{V}(c)$ nor $W' = \mathcal{V}(g) \cup \mathcal{V}(d) \cup \mathcal{V}(c')$. The proof is by case on the position of m.

1. m is a non-variable occurrence in g:
   Since $\mu\sigma(g) = t$, $\mu\sigma(g)_{|m} = t_{|m} =_T \mu\sigma(l)$, $\mu\sigma$ is solution of the constraint $c_0 = c \wedge c' \wedge (g_{|m} =^?_T l)$. So there exists a constrained critical pair $(\langle p, q\rangle, c_0)$ between the two rules, namely $(\langle g[r]_m, d\rangle, c \wedge c' \wedge (g_{|m} =^?_T l))$. Now there exists $\alpha \in CSS(c \wedge c' \wedge (g_{|m} =^?_T l))$ such that $\alpha \leq^{W \cup W'}_T \mu\sigma$. This yields: $t'' = \mu\sigma(g[r]_m) =_T \beta(\alpha(p))$ and $t' = \mu\sigma(d) =_T \beta(\alpha(q))$.
2. $m = n.p$ with n being an occurrence of a variable x in g, $(x =^?_\emptyset t_0) \in c'\downarrow$ and p being an occurrence of t_0:
   since $\mu\sigma$ is solution of $c \wedge c' \wedge (g =^?_\emptyset t) \wedge (l =^?_T t_{|m})$, $\mu\sigma(g) = \mu\sigma(t)$, $\mu\sigma(g)_{|m} = \mu\sigma(t_{|m}) = \mu\sigma(t_{0|p}) =_T \mu\sigma(l)$. So the constraint $(c \wedge c' \wedge (t_{0|p} =^?_T l))$ is satisfiable and there exists a critical pair in constraints between the two rules, namely $(\langle g, d\rangle, c_0)$ with $c_0 = c \wedge (c'\downarrow - (x =^?_\emptyset t_0)) \wedge (x =^?_\emptyset t_0[r]_p) \wedge (t_{0|p} =^?_T l)$. Now $t' = \mu\sigma(d) \xrightarrow{*}^{\mathcal{S}(R),T} \sigma'(d)$ and $t'' \xrightarrow{*}^{\mathcal{S}(R),T} \sigma'(g)$ where σ' is a solution of c_0. Then there exists $\alpha \in CSS(c_0)$ such that $\alpha \leq^{W \cup W'}_T \sigma'$. Thus we get: $t'' \xrightarrow{*}^{\mathcal{S}(R),T} =_T \beta(\alpha(g)) = \beta(\alpha(p))$ and $t' \xrightarrow{*}^{\mathcal{S}(R),T} =_T \beta(\alpha(d)) = \beta(\alpha(q))$.
3. m does not belong to $\sigma(g)$:
   then there exists a variable y at some occurrence n of $\sigma(g)$ and a subterm $u = \mu(y)$ of t, such that $u \rightarrow^{\mathcal{S}(R),T} u'$ using $(\sigma(l) \rightarrow \sigma(r))$. Let μ' be the substitution defined by $\mu'(z) = \mu(z)$ if $z \neq y$ and $\mu'(y) = u'_0$. Then $t'' \xrightarrow{*}^{\mathcal{S}(R),T} \mu'\sigma(g)$ and $t' \xrightarrow{*}^{\mathcal{S}(R),T} \mu'\sigma(d)$. Eventually $\mu'\sigma(g) \longrightarrow^{\mathcal{S}(R),T} \mu'\sigma(d)$.

Let consider now the case of a constrained cliff. If the constrained rewriting and replacement apply on disjoint subterms of t, then they simply commute. Otherwise, we only need to consider the case where the $\longleftarrow$-rewrite step modulo T occurs below the $\longleftrightarrow$-rewrite step. Without lost of generality, we can assume that $(g = d)$ applies at occurrence ϵ in t, so $t = \mu(g)$.

The proof depends on the occurrence m of the rewriting.

1. m is a non-variable occurrence in g:
   Since $\mu(g) = t$, $\mu(g)_{|m} = t_{|m} =_T \mu\sigma(l)$, $\mu\sigma$ is solution of $(c \wedge (l =^?_T g_{|m}))$. So there exists a constrained extended rule $g[l]_m \rightarrow g[r]_m$, $c \wedge (l =^?_T g_{|m})$. Since $\mu\sigma$ is solution of $c \wedge (l =^?_T g_{|m})$, there exists $\alpha \in CSS(c \wedge (l =^?_T g_{|m}))$ such that $\alpha \leq^{W \cup W'}_T \mu\sigma$ where $W = \mathcal{V}(l) \cup \mathcal{V}(r) \cup \mathcal{V}(c)$ and $W' = \mathcal{V}(g) \cup \mathcal{V}(d)$. This yields: $t'' =_T \beta(\alpha(g[r]_m))$ and $t' =_T \beta(\alpha(g[l]_m))$.
2. m does not belong to the non-variable occurremces of g:
   then there exists a variable y at some occurrence n of g such that $u = \mu(y) \rightarrow^{\mathcal{S}(R),T} u'$ using $(\sigma(l) \rightarrow \sigma(r))$. Let μ' be the substitution defined by $\mu'(z) = \mu(z)$ if $z \neq y$ and $\mu'(y) = u'$. Then $t'' \xrightarrow{*}^{\mathcal{S}(R),T} \mu'(g)$, $t' \xrightarrow{*}^{\mathcal{S}(R),T} \mu'(d)$ and $\mu'(g) =_T \mu'(d)$. □

9.6 Completion

Let CP be a set of constrained equalities denoted $(p = q, c)$, CR the current set of constrained rules, CE the current set of constrained extended rules. R will denote $CR \cup CE$. We assume that P contains $(q = p, c)$ whenever it contains $(p = q, c)$. Since we are considering equational completion through the constraint formalism, we must take care of the application of the collapse inference rule [Bac87] and we need a well-founded ordering on constrained rules denoted hereafter by $\rhd$. It can be deduced from any well-founded ordering on rules $\rhd_0$ (for instance the specialization ordering used in [Bac87]) by defining: $(l \to r, c) \rhd (l' \to r', c')$ if $\forall \sigma \in Sol(c), \exists \sigma' \in Sol(c')$ such that $\sigma(l \to r) \rhd_0 \sigma'(l' \to r')$.

Let $CCP(R)$ denote the set of constrained critical pairs of R, $CCCP(R)$ the set of its critical pairs in constraints, and $EXT(R)$ the set of its constrained extended rules.

A *constrained completion procedure modulo* T is expressed by the set **CC** of inference rules described in Figure 9.6:

Note that each of these inference rules represents a family of inference rules of the completion modulo T. This is a consequence of the Propositions 8, 9, 10 and 11.

Each rule allows computing $(CP_{i+1}, CR_{i+1}, CE_{i+1})$ from (CP_i, CR_i, CE_i), which is denoted by

$$(CP_i, CR_i, CE_i) \vdash (CP_{i+1}, CR_{i+1}, CE_{i+1}).$$

9.7 Proof of the constrained completion process

The correctness proof is a consequence of the correctness of the constrained deduction rules given in Section 4. However a more direct proof can be given.

Proposition 12 *Each inference rule of the contrained completion* **CC** *is correct, i.e.* $=_{\mathcal{S}(CP_i \cup CR_i \cup CE_i) \cup T}$ *and* $=_{\mathcal{S}(CP_{i+1} \cup CR_{i+1} \cup CE_{i+1}) \cup T}$ *are equal on* $\mathcal{T}(F)$.

Proof: For each inference rule, one can check that the two congruences coincide. □

Let CR_* be the set of all generated constrained rules, CE_* the set of all generated constrained extended rules, and CP_* the set of all generated constrained equalities.

Let CR_∞, CE_∞ and CP_∞ be respectively the set of persisting constrained rules, extended rules and equalities, i.e. the sets effectively generated by completion starting from (CP_0, CR_0, CE_0). Formally

$$CP_\infty = \bigcup_i \bigcap_{j>i} CP_j, \quad CR_\infty = \bigcup_i \bigcap_{j>i} CR_j$$
$$CE_\infty = \bigcup_i \bigcap_{j>i} CE_j, \quad R_\infty = CR_\infty \cup CE_\infty.$$

The set of rules schematized by R_∞ is denoted by $\mathcal{S}(R_\infty)$.

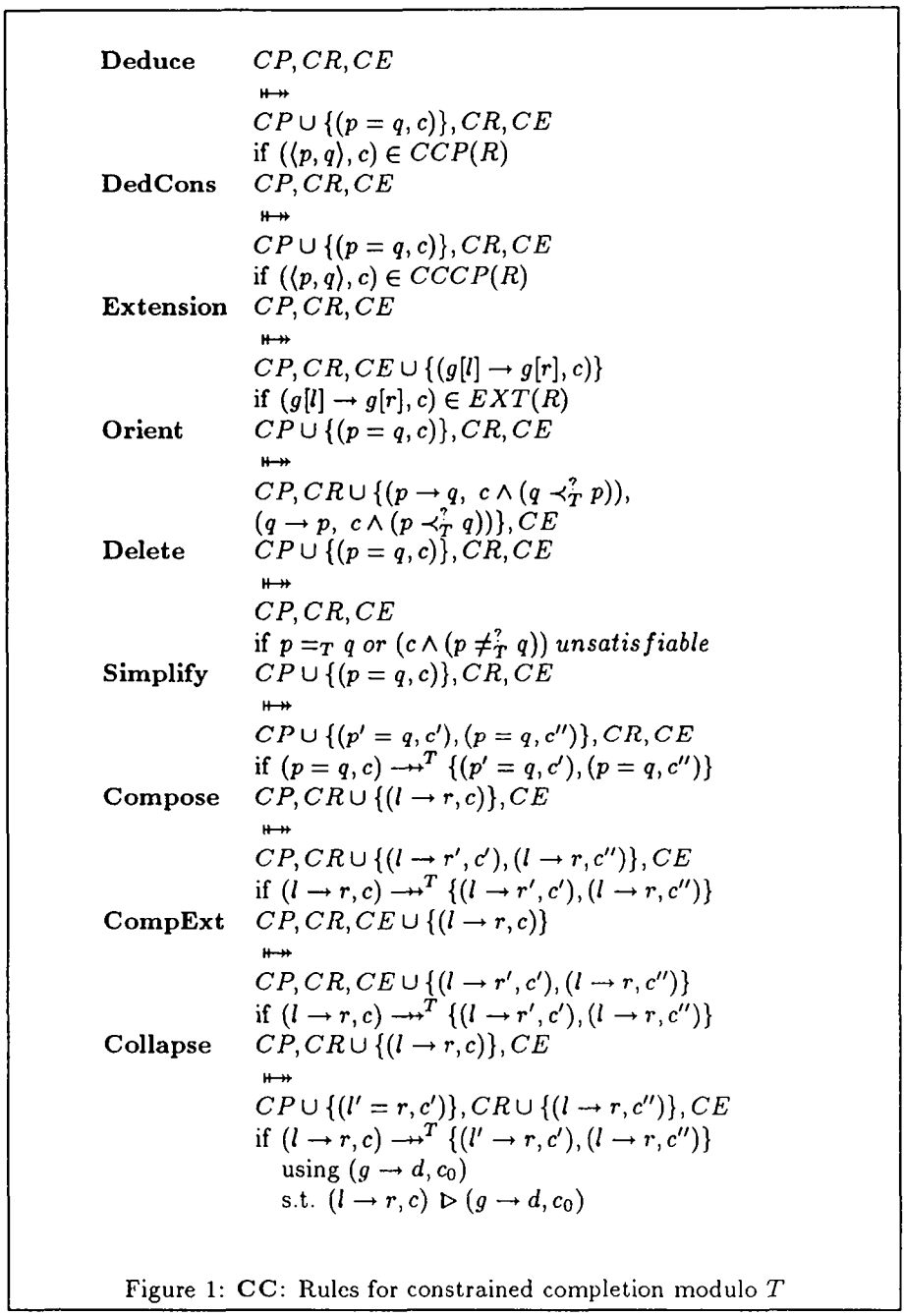

Figure 1: **CC**: Rules for constrained completion modulo T

Definition 25 *A derivation* $(CP_0, CR_0, CE_0) \vdash (CP_1, CR_1, CE_1) \vdash \ldots$ *is* fair *if*

- *the set of all constrained critical pairs and critical pairs in constraints of* R_∞, $CCP(R_\infty) \cup CCCP(R_\infty)$, *is a subset of* CP_*,
- *and the set of all constrained extended rules of* R_∞, $EXT(R_\infty)$, *is a subset of* CE_*.

The following result states the completeness of constrained completion for equational logic.

Theorem 2 *Let* $\succ_T$ *be a reduction ordering compatible with* T *and total on equivalence classes modulo* T. *Let* CP_0 *be a set of equalities with the trivial equational problem* $\top$ *as constraint. If the derivation* $(CP_0, CR_0 = \emptyset, CE_0 = \emptyset) \vdash (CP_1, CR_1, CE_1) \vdash \ldots.$ *is fair, then any ground equational theorem* $(t = t')$ *using* P_0 *has a rewrite proof using the class rewrite system* $(\mathcal{S}(R_\infty), T)$.

Proof: (Sketch) Let us consider any ground provable equality $(t = t')$ with the initial unconstrained set of equalities $P_0 \cup T$. To each (CP_i, CR_i, CE_i) is associated a proof of $(t = t')$ using constrained rules in $R_i = CR_i \cup CE_i$, constrained equalities in CP_i and axioms in T. Following [Jou87, Bac87], to each inference rule is associated a transformation rule on proofs using $(\mathcal{S}(CP_* \cup CR_* \cup CE_*) \cup T)$. For instance , to the rule **Deduce** is associated a transformation rule

$$t'' \longleftarrow^{\alpha(l)\to\alpha(r),T} t \longrightarrow^{\beta(g)\to\beta(d),T} t'$$
$$\Longrightarrow$$
$$t'' \longrightarrow^{*}{}^{\mathcal{S}(R),T} =_T \beta(\alpha(p)) \longleftrightarrow^{\alpha(p)=\alpha(q)} \beta(\alpha(q)) =_T \longleftarrow^{*}{}^{\mathcal{S}(R),T} t'.$$

Proving that the relation $\Longrightarrow$ induced by these transformation rules is terminating is achieved by finding a complexity measure $c(\mathcal{P})$ for each proof $\mathcal{P}$ and a well-founded ordering $>$ on these measures, that satisfy the following property: $\mathcal{P} \Longrightarrow \mathcal{P}'$ implies $c(\mathcal{P}) > c(\mathcal{P}')$, for each proof transformation rule. Actually $c(\mathcal{P})$ can be defined as in [BD89].

The next step is to show that each ground proof has a rewrite proof for $(\mathcal{S}(R_\infty), T)$, by noetherian induction on $\Longrightarrow$. Assume that the proof is not a rewrite proof:

- If a non-persisting equality or a non-persisting rule is used, by definition, there exists a step i such that it belongs to $CP_i \cup R_i$ and not to $CP_{i+1} \cup R_{i+1}$. Since $\Longrightarrow$ reflects $\vdash$, the proof is reducible by $\Longrightarrow$ to a proof that does not use it any more and the induction hypothesis can be applied.
- If the proof contains a peak or a cliff, it is also reducible, due to Lemma 3 and to the fairness hypothesis. Again the induction hypothesis can be applied on the reduced proof.

- It contains no equality step using CP_∞, since every constrained pair can be ordered.

□

Thus, from the result of a fair derivation, it is possible to deduce a set of rewrite rules $\mathcal{S}(R_\infty)$ with the Church-Rosser property modulo T on ground terms.

10 Conclusion

The notion of constrained rewriting bears much similarity with conditional rewriting, especially with contextual rewriting. However, in conditional rewriting, occurrences of the same function symbol in conditions and in conclusion are usually interpreted in the same way. This is no more true with constrained rewriting, where the symbols in constraints are subject to special deduction rules. For instance, an equation $(f(s)=^?_\emptyset f(t))$ in a constraint may be decomposed into $(s=^?_\emptyset t)$. Such a transformation is in general not valid in the first-order theory which underlies the constrained formula. The difference between constrained and conditional rewriting also appears for instance in the following example of idempotent semigroups from [SS82]. It is a noetherian confluent conditional system for the theory of an idempotent associative symbol $*$.

$$x * x \longrightarrow x$$
$$x * y * z \longrightarrow x * z \text{ if } (x=^?_{ACI} z) \wedge (x * y=^?_{ACI} z)$$

where ACI is the theory of an associative commutative idempotent symbol. So the equations in the condition are solved modulo the theory ACI of $*$, while the rules are used with matching modulo associativity. Since the theory of $*$ is different in the constraints, due to the commutativity axiom, we feel that this system is typically a constrained rewrite system. Actually this system does not fit into the classical frameworks for conditional term rewriting [KR89].

The notion of constrained resolution has already been studied for equational constraints in the empty theory. Caferra and Zabel [CZ90] use it in a procedure which is able to refute or to generate some kind of models when they exist. Buntine and Bürckert [BB89] also noticed that constrained resolution improve the efficiency of theorem provers, for instance by preventing from the generation of tautologies, an idea further developed in [Bur90].

We have shown how constraints help to describe deduction procedures in a very precise way, and as a consequence, to improve drastically their efficiency. Several examples have been developed in equational or first-order logic. While previous related works have always been oriented towards specific applications, we have been motivated here by giving a general setting for reasoning with symbolic constraints.

Future work should be devoted to the design of efficient constraint satisfiability checkers and constraint solvers but also to handling new kind of constraints.

Typing constraints are particularly essential for building new programming languages and provers. Numerical constraints are also worth investigating in this framework. Beside their natural interest, they may appear for instance as polynomial constraints in orientation problems.

Last, a special investigation effort has to be put on the design of strategies for managing deduction rules in an efficient way. We think that expressing all the deduction rules, both on constraints and on formulae, in a uniform formalism of transformation rules, is a helpful step towards this goal.

Acknowledgements: We especially thank H. Comon and E. Domenjoud for fruitful discussions on the subjects of this paper, and U. Waldmann for his valuable comments.

References

[Bac87] L. Bachmair. *Proof methods for equational theories.* PhD thesis, University of Illinois, Urbana-Champaign, 1987. Revised version, August 1988.

[BB89] W. Buntine and H.-J. Bürckert. *On Solving Equations and Disequations.* Technical Report SR-89-03, Universität Kaiserslautern, 1989.

[BD89] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2-3):173–202, October 1989.

[BDP89] L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 1–30, Academic Press, 1989.

[Bur87] H.-J. Bürckert. *Solving Disequations in Equational Theories.* Technical Report SEKI-Report SR-87-15, Universität Kaiserslautern, 1987.

[Bur90] H.-J. Bürckert. A resolution principle for clauses with constraints. In M. Stickel, editor, *Proceedings of CADE'90*, pages 178–192, Springer-Verlag, Lecture Notes in Computer Science, Kaiserslautern, July 1990.

[CL89] H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7(3 & 4):371–426, 1989. Special issue on unification. Part one.

[Com88] H. Comon. Unification et disunification. Théories et applications. Thèse de l'Institut Polytechnique de Grenoble, 1988.

[Com90a] H. Comon. Equational formulas in order-sorted algebras. In *Proceedings ICALP'90*, 1990.

[Com90b] H. Comon. Solving inequations in term algebras. In *Proceedings of the 5th IEEE Symposium on Logic in Computer Science*, Philadelphia, June 1990.

[CZ90] R. Caferra and N. Zabel. A method for simultaneous search for refutations and models using equational problems. 1990. submitted.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, chapter 15: Rewrite systems. Volume B, North-Holland, 1990. Also as: Research report 478, LRI.

[Fay79] M. Fay. First order unification in equational theories. In *Proceedings 4th Conference on Automated Deduction*, pages 161–167, Springer-Verlag, Austin (Texas), 1979. Lecture Notes in Computer Science, volume 87.

[Gal86] J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving.* Volume 5 of *Computer Science and Technology Series*, Harper & Row, New-York, 1986.

[Gra79] G. Grätzer. *Universal Algebra.* Springer-Verlag, 1979. Second Edition.

[HJM*87] N. Heintze, J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. *The CLP(R) Programmer's Manual.* Monash University, Clayton 3168, Victoria, Australia., 1987.

[HO80] G. Huet and D. Oppen. Equations and rewrite rules: a survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405, Academic Press, New-York, 1980.

[HR86] J. Hsiang and M. Rusinowitch. A new method for establishing refutational completeness in theorem proving. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction*, pages 141–152, Springer-Verlag, 1986. Lecture Notes in Computer Science, volume 230.

[HR87] J. Hsiang and M. Rusinowitch. On word problem in equational theories. In Th. Ottmann, editor, *Proceedings of 14th International Colloquium on Automata, Languages and Programming, Karlsruhe (West Germany)*, Springer-Verlag, July 1987. Lecture Notes in Computer Science, volume 267.

[Hue76] G. Huet. Résolution d'equations dans les langages d'ordre 1,2, ...,ω. Thèse d'état de l'Université de Paris 7, 1976.

[Hul80] J.-M. Hullot. Canonical forms and unification. In W. Bibel and R. Kowalski, editors, *Proceedings of 5th Conference on Automated Deduction*, pages 318–334, Springer-Verlag, July 1980. Lecture Notes in Computer Science, volume 87.

[JK86] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986. Preliminary version in Proceedings 11th ACM Symposium on Principles of Programming Languages, Salt Lake City, 1984.

[JK90] J.-P. Jouannaud and C. Kirchner. *Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification.* Research Report, CRIN, 1990. To appear in *Festschrift for Robinson*, J.-L. Lassez and G. Plotkin Editors, MIT Press.

[JKK83] J.-P. Jouannaud, C. Kirchner, and H. Kirchner. Incremental construction of unification algorithms in equational theories. In *Proceedings International Colloquium on Automata, Languages and Programming*, pages 361–373, Springer-Verlag, Barcelona (Spain), 1983. Lecture Notes in Computer Science, volume 154.

[JL86] J. Jaffar and J.L. Lassez. *Constraint Logic Programming.* Technical Report, Monash University Australia, June 1986.

[Jou87] J.-P. Jouannaud. Proof algebras. *Invited lecture at the second Rewriting Techniques and Applications Conference, Bordeaux (France)*, 1987.

[KB70] 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, Oxford, 1970.

[Kir85] C. Kirchner. Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles. Thèse d'état de l'Université de Nancy I, 1985.

[Kir88] C. Kirchner. Order-sorted equational unification. Presented at the fifth International Conference on Logic Programming (Seattle, USA), August 1988. Also as Rapport de Recherche INRIA 954, Dec 88.

[KK89] C. Kirchner and H. Kirchner. Constrained equational reasoning. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, pages 382–389, ACM Press, Portland (Oregon), July 1989. Report CRIN 89-R-220.

[KL87] C. Kirchner and P. Lescanne. Solving disequations. In D. Gries, editor, *Proceeding of the Second Symposium on Logic In Computer Science*, pages 347–352, IEEE, 1987.

[KR89] S. Kaplan and J.-L. Rémy. Completion algorithms for conditional rewriting systems. In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 141–170, Academic Press, 1989.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions On Programming Languages And Systems*, 4(2):258–282, 1982.

[MN90] U. Martin and T. Nipkow. Ordered rewriting and confluence. In *Proceedings of CADE'90*, Springer-Verlag, Kaiserslautern (RFA), 1990. Lecture Notes in Computer Science.

[Pet83] G. Peterson. A technique for establishing completeness results in theorem proving with equality. *SIAM Journal of Computing*, 12(1):82–100, 1983.

[Pet90] G.E. Peterson. Complete sets of reductions with constraints. In *Proceedings 10th International Conference on Automated Deduction*, Springer-Verlag, Kaiserslautern (RFA), 1990. Lecture Notes in Computer Science.

[PS81] G. Peterson and M. Stickel. Complete sets of reductions for some equational theories. *Journal of the Association for Computing Machinery*, 28:233–264, 1981.

[Rob65] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12, 1965.

[Rus88] M. Rusinowitch. Theorem-proving with resolution and superposition: an extension of Knuth and Bendix procedure to a complete set of inference rules. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1988.

[Rus89] M. Rusinowitch. *Démonstration automatique-Techniques de réécriture*. InterEditions, 1989.

[Sch87] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations.* PhD thesis, Universität Kaiserslautern (West Germany), 1987.

[Smo89] G. Smolka. *Logic Programming over Polymorphically Order-Sorted Types.* PhD thesis, Universität Kaiserslautern, FB Informatik, West-Germany, 1989.

[SS82] J. Siekmann and P Szabó. A noetherian and confluent rewrite system for idempotent semigroups. *Semigroup Forum*, 25:83–110, 1982.

[Tay79] W. Taylor. Equational logic. *Houston Journal of Mathematics*, 1979. Appears also in [Gra79], Appendix 4.

[vZGS78] J. von Zur Gathen and M. Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, October 1978.

Contents

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399