Hubert Comon, Claude Marché, Ralf Treinen

Constraints in Computational Logics, International Summer School, September 5–8, 1999, Gif-sur-Yvette, France, Edition

SPIN Springer's internal project number, if known

Springer

Berlin Heidelberg New York
Barcelona Budapest Hong Kong
London Milan Paris
Santa Clara Singapore Tokyo

Contents

Preface

CCL (*Construction of Computational Logics* and later *Constraints in Computational Logic*) is the name of an ESPRIT working group which met regularly from 1992 to 1999 (see `http://www.ps.uni-sb.de/ccl/`). It gathered together research teams from Germany, France, Spain and Israel, managed by the company COSYTEC.

In the last years, the main emphasis of the working group was on *constraints*, techniques to solve them, combine them and applications ranging from industrial applications to logic programming and automated deduction. At the end of the working group, during the fall 1999, we organized a summer school, intending to summarize the main advances performed in the field during the last 7 years. The present book contains the (revised) lecture notes of this school. It contains six chapters, each of which written by some member(s) of the working group and covering the various aspects of constraints in computational logic. We intended it to be read by non specialists, though a prior knowledge in first-order logic and programming is probably necessary.

Constraints provide with a declarative way of representing infinite sets of data. As we demonstrate (hopefully) in this book, they are well suited for the *combination* of different logical or programming paradigms. This is known since the 80's for *constraint logic programming*, but has been combined with functional programming in more recent years; a chapter (by M. Rodríguez-Artalejo) is devoted to the combination of constraints, logic and functional programming.

The use of constraints in automated deduction is more recent and turned out to be very successful, moving the control from the meta-level to the constraints, which are now first-class objects. This allows to keep an history of the reasons why deductions were possible, hence restricting further deductions. A chapter of this book (by H. Ganzinger and R. Nieuwenhuis) is devoted to constraints and theorem proving.

Constraints are not only a nice mathematical construction. The chapter (by H. Simonis) on industrial applications shows the important recent development of constraint solving in real life applications, for instance scheduling, decision making and optimization.

Combining constraints (or *combining decision procedures*) emerged during the last years as an important issue in theorem proving and verification; constraints turn out to be an adequate formalism to combine efficient techniques on each particular domain, getting algorithms for mixed domains. There is now a biannual workshop on these topics, whose proceedings are published in the LNAI series. The chapter on Combining Constraints Solving (by F. Baader and K. Schulz) introduces the subject and surveys the results.

Before these four introductory chapters on applications of constraint solving, the introductory chapter (by J.-P. Jouannaud and R. Treinen) provides with a general introduction to constraint solving and the chapter on constraint solving on terms (by H. Comon and C. Kirchner) introduces the constraint solving techniques which are used in, e.g. applications to automated deduction.

Every chapter includes an important bibliography, to which the reader is referred for more informations.

Orsay and Stanford,
December 2000

Hubert Comon
Claude Marché
Ralf Treinen

1 Constraints and Theorem Proving

Harald Ganzinger[1] and Robert Nieuwenhuis[2] *

[1] Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany. Email: hg@mpi-sb.mpg.de
[2] Technical University of Catalonia, Jordi Girona 1, 08034 Barcelona, Spain. Email: roberto@lsi.upc.es

1.1 Introduction

Symbolic constraints provide a convenient way of describing the connection between deduction at the level of ground formulae and the general inferences as they are computed by a prover on formulae with variables.

For instance, each resolution inference represents infinitely many ground inferences, obtained by substituting ground terms for the variables in the clauses. For refutational completeness only certain restricted cases of ground inferences are required. In the case of ordered resolution, essential ground inferences only resolve maximal literals. On the non-ground level, these ordering restrictions can be represented as a kind of restricted quantification: only ground terms that satisfy the ordering restrictions should be substituted for the variables. This leads us to resolution calculi with order-constrained clauses, where in each inference the conclusion is restricted by the ordering constraints of the current deduction step, combined with the constraints that are already present in the premises. Clauses with unsatisfiable constraints represent no ground clauses and can hence be removed during the deduction process, thus pruning the search space.

In these notes we mainly focus on saturation-based theorem proving methods including resolution, superposition and chaining. In this context, apart from ordering restrictions, constraints are also used to describe unification problems. This paper consist of three main parts.

First we introduce the main concepts for the case of pure equational logic, where rewriting and Knuth/Bendix completion are fundamental deduction methods. The topics which we cover include Birkhoff's theorem, first-order vs. inductive consequences, term rewrite systems, confluence, termination, orderings and critical pair computation (i.e., superposition).

In the second part, we extend these ideas to saturation-based methods for general first-order clauses, including superposition and chaining-based inference systems, completeness proofs by means of the model generation method and an abstract notion of redundancy for inferences.

The third part covers several extensions, with special attention to those aspects for which constraints play an important role, like basic strategies,

* Both authors supported by the ESPRIT working group CCL-II, ref. WG # 22457.

deduction modulo equational theories, constraint-based redundancy methods, and deduction-based complexity analysis and decision procedures.

For more details and further references the reader is referred to [BG98] and [NR01].

1.2 Equality clauses

We briefly recall some basic concepts on first-order logic with equality. In particular, we give the syntax and semantics for the fragment of clausal (and hence only universally quantified) formulae. We refer the reader to standard textbooks such as [Fit90] for clausal normal form transformations of arbitrarily quantified first-order formulae.

1.2.1 Syntax: terms, atoms, equations, and clauses

Let $\mathcal{F}$ be a set of *function symbols*, $\mathcal{X}$ an infinite set of *variables* (disjoint with $\mathcal{F}$), and a function *arity*: $\mathcal{F} \rightarrow \mathbb{N}$ associating an *arity* (a natural number) to each function symbol. Function symbols f with $arity(f) = n$ are called *n-ary* symbols. In particular, f is called *unary*, if $n = 1$, *binary*, if $n = 2$, and a *constant symbol*, if $n = 0$.

$T(\mathcal{F}, \mathcal{X})$ denotes the set of *terms with variables* or simply *terms*: a term is a variable or an expression $f(t_1, \ldots, t_n)$ where f is an n-ary function symbol and the t_i are terms for $1 \leq i \leq n$.

Note that terms can be seen as ordered trees where the leaves are labeled by constant symbols or variables, and where every other node is labeled by a function symbol of arity at least 1. The nodes in terms can be addressed by *positions* which are sequences of natural numbers. The root in a term is addressed by position λ (denoting the empty sequence) and if p is the address of a node v in t having the $n \geq 0$ successors $v_1, \ldots, v_n$, then these have addresses $p.1, \ldots, p.n$, respectively. We say that a function symbol or a variable *occurs* (at position p) in a term t if the node addressed by p carries that symbol as a label. By $vars(t)$ we denote the set of all variables occurring in t.

Let $\mathcal{P}$ be a set of *predicate symbols* with arity (disjoint from $\mathcal{F}$ and $\mathcal{X}$). Then $p(t_1, \ldots, t_n)$ is an *atom* if p is an n-ary predicate symbol and the t_i are terms for $1 \leq i \leq n$. In the sequel it will often simplify matters technically to also view atoms as terms of a specific sort which, in particular, cannot occur as proper subterms of other terms.

By $t|_p$ we denote the *subterm of t at position p*: we have $t|_\lambda = t$, and $f(t_1, \ldots, t_n)|_{i.p} = t_i|_p$ if $1 \leq i \leq n$ (and is undefined if $i > n$). We also write $t[s]_p$ to denote the term obtained by replacing in t the subterm at position p by the term s. For example, if t is $f(a, g(b, h(c)), d)$, then $t|_{2.2.1} = c$, and $t[d]_{2.2} = f(a, g(b, d), d)$.

A *multiset* over a set S is a function $M\colon S \to I\!N$. The union of multisets is defined by $M_1 \cup M_2(x) = M_1(x) + M_2(x)$. We sometimes also use a set-like notation: $M = \{a, a, b\}$ denotes the multiset M where $M(a) = 2$, $M(b) = 1$, and $M(x) = 0$ for $x \neq a$ and $x \neq b$.

A first-order *clause* is a pair of finite multisets of atoms (Γ, Δ), written $\Gamma \to \Delta$, where Γ is called the *antecedent*, and Δ the *succedent* of the clause. The *empty clause* $\Box$ is a clause where both Γ and Δ are empty, and a *Horn clause* is a clause whose succedent Δ contains at most one atom. In clauses we often use comma to denote the union of multisets or the inclusion of an atom in a multiset; for example, we write $A, \Gamma, \Gamma' \to \Delta$ instead of $\{A\} \cup \Gamma \cup \Gamma' \to \Delta$. A literal is either an atom (a *positive literal*) or a negation thereof (a *negative literal*). Clauses $A_1, \ldots, A_n \to B_1, \ldots, B_k$ can also be written as a disjunction of literals $\neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_k$. Hence, the A_i are called the *negative* atoms, and the B_j the *positive* atoms, respectively, of C.

1.2.2 Semantics

An *interpretation* I for $\mathcal{F}$ and $\mathcal{P}$ consists of three components:

1. a non-empty set D, called the *domain*,
2. for each n-ary f in $\mathcal{F}$, a function $f_I\colon D^n \to D$, and
3. for each n-ary p in $\mathcal{P}$, a function $p_I\colon D^n \to \{\mathit{true}, \mathit{false}\}$

Any interpretation I is a mathematical model of some intended real-world meaning of the symbols in the vocabulary. Variables denote elements in the domain D. This is formalized by the notion of an *assignment* which is a function $as\colon \mathcal{X} \to D$. A clause C is *true* in I, written $I \models C$, if for every assignment as, at least one of its negative atoms evaluates to *false*, or at least one of its positive atoms evaluates to *true*, according to the following evaluation function *eval*:

$$\begin{array}{lll} \mathit{eval}(p(t_1, \ldots, t_n)) & = p_I(\mathit{eval}(t_1), \ldots, \mathit{eval}(t_n)) & \text{if } p \in \mathcal{P} \\ \mathit{eval}(f(t_1, \ldots, t_n)) & = f_I(\mathit{eval}(t_1), \ldots, \mathit{eval}(t_n)) & \text{if } f \in \mathcal{F} \\ \mathit{eval}(x) & = as(x) & \text{if } x \in \mathcal{X} \end{array}$$

Instead of saying that C is true in I, we also say that I *satisfies* C or that I *is a model of* C. A set of clauses S is true in an interpretation I if every clause of S is true in I. From these stipulations we see that the variables in clauses are implicitly universally quantified.

Propositional logic arises as the particular case when all predicate symbols have arity zero, and hence the formulae contain no function symbols or variables. Then an interpretation just says which predicate symbols are *true* and which ones are *false*.

1.2.3 Why is satisfiability so interesting?

Let us now consider a number of relevant questions that appear in practical applications of (any) logic. If we describe a certain real-world problem such as a circuit, a communication protocol, a data base by a logical formula F, we may ask ourselves:

1. whether F has no inherent contradictions; this amounts to whether there exists an interpretation I such that $I \models F$, i.e, whether F is *satisfiable*.
2. whether F is true in all situations; this amounts to the question whether $I \models F$ for every interpretation I, i.e., whether F is a *tautology*.
3. whether another formula G holds in all the real-world situations described by F; this amounts to the question whether $I \models G$ whenever $I \models F$, for every interpretation I, i.e., whether G is a *logical consequence* of F. In this case we also simply say that G *follows from* F, and we write $F \models G$ (overloading the symbol $\models$).
4. whether another (perhaps simpler in some sense) formula G describes exactly the same real-world situations as F; this amounts to the question whether $I \models G$ if, and only if, $I \models F$, for every interpretation I, i.e., whether F and G are *logically equivalent*.

Here we concentrate on the satisfiability problem for sets of clauses, since all the aforementioned questions (for full first-order logic) can be reduced to it: F is a tautology iff $\neg F$ is unsatisfiable, $F \models G$ iff $F \wedge \neg G$ is unsatisfiable. F and G are logically equivalent iff $(F \wedge \neg G) \vee (G \wedge \neg F)$ is unsatisfiable. Also, numerous satisfiability-preserving transformations of arbitrary formulae F into sets of clauses S have been described in the literature (but such transformations are out of the scope of this course).

For propositional logic, the satisfiability problem of a finite clause set can be decided as follows. If the cardinality of $\mathcal{P}$ is n, then there are 2^n possible interpretations (the rows of the so-called truth-tables). For each interpretation the formulae can be evaluated. Less naive algorithms are used in practice (but the problem is well-known to be NP-complete, and hence no polynomial algorithm is believed to exist).

1.2.4 Herbrand interpretations

It is well-known that the satisfiability problem for sets of clauses S is undecidable, that is, there is no algorithm that takes as input any clause set S and that always terminates producing a correct yes/no answer regarding the satisfiability of S. So what can we do if we want to (try to) find out whether or not a given clause set S is satisfiable? It is certainly not a good idea to just generate interpretations and, for each one of them, try to check whether it satisfies S. One of the reasons is that the set of interpretations to be tried is large, usually infinite. Fortunately, as we will explain now, it suffices to consider a much smaller (although still not finite) set of interpretations, the so-called *Herbrand* interpretations: we will see that a set of clauses S is satisfiable if, and only if, it has a Herbrand model. Both the domain and the interpretation of the function symbols are fixed by definition in Herbrand interpretations:

Let $T(\mathcal{F})$ denote the set of *ground* terms (also sometimes called *closed* terms) consists of all terms without variables, that is, the terms $f(t_1, \ldots, t_n)$, where f is an n-ary function symbol and the t_i are ground terms for $1 \leq i \leq n$.

A *Herbrand* interpretation is an interpretation with domain $T(\mathcal{F})$, and where $f_I(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for every n-ary function symbol f and ground terms $t_1, \ldots, t_n$.

(Some people find this definition a bit tricky: the rather similar expressions $f_I(t_1, \ldots, t_n)$ and $f(t_1, \ldots, t_n)$ have completely different meanings; we are saying that the function f_I applied to the terms $t_1, \ldots, t_n$ gives as result the term $f(t_1, \ldots, t_n)$. In Herbrand interpretations, functions are interpreted as term constructors.) Note that two Herbrand interpretations for given $\mathcal{F}$ and $\mathcal{P}$ can only differ in the interpretation of the predicate symbols. Therefore it is appropriate to identify any Herbrand interpretation I with the set of ground atoms that are true in I. Under this view, a ground clause $\Gamma \rightarrow \Delta$ is true in I if either $\Gamma \not\subseteq I$, or else $\Delta \cap I \neq \emptyset$.

It is not difficult to see that a set of clauses S is satisfiable if, and only if, it has a Herbrand model. For the non-trivial implication, if $I \models S$, we can construct the Herbrand model I' as the set of all ground atoms $p(t_1, \ldots, t_n)$ for which $p_I(eval(t_1), \ldots, eval(t_n)) = true$, where *eval* is the evaluation function associated with I. The details are left as an exercise to the reader.

This result is the basis for many deduction methods in first-order logic: in order to show unsatisfiability, it suffices to show that there is no Herbrand model.

1.2.5 Herbrand's theorem

A *substitution* σ is a mapping from variables to terms. We sometimes write substitutions as a set of pairs $x \mapsto t$, where x is a variable and t is a term. For example, if $\sigma(x) = a$ and $\sigma(y) = b$ and $\sigma(z) = z$ for all $z \in \mathcal{X} \setminus \{x, y\}$, then we write $\sigma = \{x \mapsto a, y \mapsto b\}$. The *domain* of a substitution σ is the set of variables x for which $\sigma(x) \neq x$. We require the domain of substitutions to be finite. A substitution can be extended to a function from terms (atoms, clauses) to terms (atoms, clauses) as follows: using a postfix notation, $t\sigma$ denotes the result of simultaneously replacing in t every $x \in Dom(\sigma)$ by $x\sigma$. For example, $f(x, y)\{x \mapsto g(y), y \mapsto b\} = f(g(y), b)$. A substitution σ is *ground* if its range is $T(\mathcal{F})$.

A *ground instance* of a clause C is a ground clause $C\sigma$ for some ground substitution σ. The set of ground instances $gnd(S)$ of a set of clauses S is the union of the sets of ground instances of the individual clauses in S. $gnd(S)$ is also called the *Herbrand expansion* of S. Ground atoms are not essentially different from propositional variables. Different ground atoms have no semantic connections between them so that their truth values can be chosen independently in any interpretation. Therefore propositional (clause) logic also arises as the restriction of first-order clause logic to the variable-free case.

Now there are several important results to be mentioned. The first one (Gödel-Herbrand-Skolem), states that S is satisfiable if, and only if, $gnd(S)$ is satisfiable. The reason is that, as we have seen, it suffices to consider Herbrand interpretations, where the variables range over the domain of ground terms, i.e, every assignment as to be tried is a ground substitution. This result essentially reduces the first-order satisfiability problem to the satisfiability problem for (in general, infinite) sets of propositional clauses. By the *compactness theorem* for propositional logic, an infinite set S of propositional clauses is satisfiable if, and only if, every finite subset of S is satisfiable. From this, we finally obtain Herbrand's theorem: a set of clauses S is unsatisfiable if, and only if, there is an unsatisfiable finite subset of $gnd(S)$.

This gives us a semi-decision procedure for the unsatisfiability problem, that is, a procedure that takes as input a set of clauses S and that (i) terminates with output "yes" if S is unsatisfiable, and (ii) terminates with output "no" or does not terminate otherwise. This procedure just enumerates $gnd(S)$, and from time to time checks whether the current part is satisfiable in propositional logic.

Compactness and Herbrand's theorem are the key to many classical completeness results for refutation calculi including resolution. Here, we will take a different route in that our completeness proofs for resolution and paramodulation will *not* require clause sets to be finite, hence Herbrand's theorem and compactness will follow as a corollary. For readers interested in direct proofs of compactness we refer to [Fit90].

1.2.6 Equality

Because of its importance in practice, the equality predicate deserves special attention in first-order logic and theorem proving. Equality can be defined within first-order logic as a binary predicate $\simeq$ (written in infix-notation) satisfying this set of clauses $\mathcal{E}$:

$$
\begin{array}{rll}
\rightarrow x \simeq x & & \textit{(reflexivity)} \\
x \simeq y \rightarrow y \simeq x & & \textit{(symmetry)} \\
x \simeq y \wedge y \simeq z \rightarrow x \simeq z & & \textit{(transitivity)} \\
x_1 \simeq y_1 \wedge \ldots \wedge x_n \simeq y_n \rightarrow f(x_1, \ldots, x_n) \simeq f(y_1, \ldots, y_n) & & \textit{(monotonicity)} \\
x_1 \simeq y_1 \wedge \ldots \wedge x_n \simeq y_n \quad & & \\
\wedge\, p(x_1, \ldots, x_n) \rightarrow p(y_1, \ldots, y_n) & & \textit{(monotonicity)}
\end{array}
$$

where the monotonicity axioms are, in fact, axiom *schemes*, and one instance is needed for each n-ary function and predicate symbol f and p, respectively. A binary relation satisfying $\mathcal{E}$ is called a *congruence*. An *equation* is an atom with $\simeq$ as its predicate. We will build-in the symmetry of equality into our notation, that is, we will use $s \simeq t$ to also denote the reverse equation $t \simeq s$. An *equality clause* is a clause in which all atoms are equations. By an *equality interpretation* we mean a interpretation in which $\simeq$ is a congruence.

Another way to deal with the equality predicate is to build it into the logic. Then we obtain *first-order logic with equality* (FOLE) where the only interpretations I under consideration are the ones where the semantics of the equality predicate is "being the same element of D", that is, where $\simeq_I$ is fixed by definition as: $\simeq_I(x, y) = \mathit{true}$ if, and only if, x and y are the same element of D. For our purposes, the view of equality as a congruence will be more appropriate. It is justified by the fact that a set of clauses S is satisfiable in FOLE if, and only if, $S \cup \mathcal{E}$ is satisfiable in FOL (without equality). The proof of this fact is left as a relatively easy exercise to the reader.

Note that FOLE is more "expressive" in the following sense. In FOLE $x \simeq a \vee x \simeq b$ expresses that models have cardinality at most two, which cannot be expressed in FOL without equality, where, if $I \models F$ and some a is in the domain D of I, one can always build another model I' with domain $D \cup \{a'\}$ where a' is a "clone" of a that behaves identically to a under all functions and predicates. (More generally, in FOL without equality, if I is a model of F, then F also has models over domains with any cardinality larger than that of I.)

From now on, unless explicitly indicated otherwise, we will restrict attention to equality interpretations. In particular, $F \models G$ will mean implication with respect to equality interpretations, that is, $I \models G$ whenever $I \models F$, for every equality interpretation I.

1.2.7 Equality Herbrand interpretations

If $\Rightarrow$ is a binary relation on a set, then $\Leftarrow$ denotes its inverse, $\Leftrightarrow$ its symmetric closure, $\Rightarrow^+$ its transitive closure and $\Rightarrow^*$ its reflexive-transitive closure. A relation $\Rightarrow$ on terms is called *monotonic* if $s \Rightarrow t$ implies $u[s]_p \Rightarrow u[t]_p$ for all terms s, t and u and positions p in u. A *congruence* on terms is a reflexive, symmetric, transitive and monotonic relation on terms. It is easy to see that congruences on terms as just defined are precisely the relations on $T(\mathcal{F})$ that satisfy $\mathcal{E}$. Therefore, by an *equality Herbrand interpretation* we mean a Herbrand interpretation in which $\simeq$ is a congruence. Considering equality as a congruence (rather than dealing with FOLE), the concept of a Herbrand interpretation remains meaningful and the related results continue to hold.

In what follows, we will consider equality to be the only predicate, since for every other predicate symbol p, (positive or negative) atoms $p(t_1 \dots t_n)$ can be expressed as (positive or negative) equations $p(t_1 \dots t_n) \simeq \top$, where $\top$ is a new special symbol, and where p is considered a function symbol rather than a predicate symbol.[1] It is easy to see that this transformation preserves satisfiability. The proof of this fact is left as an exercise to the reader. (Hint: one can "translate" the interpretations accordingly such that a ground atom is true in an equality Herbrand interpretation I iff in the Herbrand interpretation I' over the modified signature the term $p(t_1 \ldots t_n)$ is congruent to $\top$. Note, however that I and I' are not isomorphic since two ground atoms that are false in I need not be in the same congruence class of I'.) Also by a Herbrand interpretation we shall from now on always mean an equality Herbrand interpretation.

1.3 The purely equational case: rewriting and completion

Here we consider purely equational logic, that is, FOL where all formulae are equations (that is, positive unit clauses). Given a set E of equation and another equation $s \simeq t$, called a *query*, the problem whether or not $E \models s \simeq t$ is called the *word problem* for E. This problem can be attacked through reasoning using Leibniz' law for the "replacement of equals by equals": an equation $s \simeq t$ can be "applied" to a term u by replacing a subterm of the form $s\sigma$ of u by $t\sigma$.

More formally, if E is a set of equations, the relation $\Leftrightarrow_E^*$ is the smallest congruence such that $s\sigma \Leftrightarrow_E^* t\sigma$ for all $s \simeq t$ in E and all substitutions σ. Birkhoff's theorem states that the application of equations is correct and complete for solving the word problem: for every equation $s \simeq t$ and set of equations E, we have $E \models s \simeq t$ if, and only if, $s \Leftrightarrow_E^* t$. The reason is of course that $\Leftrightarrow_E^*$ is precisely the unique minimal Herbrand model of E; it exists for all E, and hence all sets of equations E are satisfiable.

However, Birkhoff's theorem does not provide a practical (or automatizable) deduction method. If, as an exercise, the reader tries to prove a simple statement in group theory like:

$$\{\ (x+y)+z \simeq x+(y+z),\ \ e+x \simeq x,\ \ i(x)+x \simeq e\ \} \models\ \ x+e \simeq x$$

it quickly becomes clear why: there are too many ways of applying the equations, and hence too many (sometimes larger and larger) terms are obtained.

One way to reduce the search space is to restrict the application of the equations, by only replacing "big" terms by "smaller" ones. Let us now formalize this idea.

First we need some more definitions on relations. If $\Rightarrow$ is a relation, we write $s \Rightarrow^! t$ if $s \Rightarrow^* t$ and there is no t' such that $t \Rightarrow t'$. Then t is called *irreducible* and a *normal form* of s with respect to $\Rightarrow$. The relation $\Rightarrow$ is

[1] To avoid meaningless expressions in which predicate symbols occur at proper subterms one should, however, adopt a two-sorted type discipline on terms in the encoding.

well-founded or *terminating* if there exists no infinite sequence $s_1 \Rightarrow s_2 \Rightarrow \ldots$ and it is *confluent* or *Church-Rosser* if the relation $\Leftarrow^* \circ \Rightarrow^*$ is contained in $\Rightarrow^* \circ \Leftarrow^*$. It is *locally confluent* if $\Leftarrow \circ \Rightarrow \; \subseteq \; \Rightarrow^* \circ \Leftarrow^*$. By Newman's lemma, terminating locally-confluent relations are confluent.

A *rewrite rule* is an ordered pair of terms (s,t), written $s \Rightarrow t$, and a set of rewrite rules R is a *term rewrite system* (TRS). The *rewrite relation* by R on $T(\mathcal{F},\mathcal{X})$, denoted $\Rightarrow_R$, is the smallest monotonic relation such that $l\sigma \Rightarrow_R r\sigma$ for all $l \Rightarrow r \in R$ and all σ, and if $s \Rightarrow_R t$ then we say that s *rewrites (in one step) into* t under R.

A TRS R is called terminating, confluent, etc. if $\Rightarrow_R$ is. A TRS R is *convergent* if it is confluent and terminating. For convergent TRS R, every term t has a unique normal form with respect to $\Rightarrow_R$, denoted by $nf_R(t)$. (Exercise: prove this; hint: use Noetherian induction on $\Rightarrow$.) For a convergent system R (where R is seen as a set of equations), $s \Leftrightarrow_R^* t$ if, and only if, $nf_R(s) = nf_R(t)$. In that case, by Birkhoff's theorem, $s \simeq t$ is a logical consequence of R if, and only if, $nf_R(s) = nf_R(t)$. Therefore, if R is finite and convergent, the word problem for R is decidable by rewriting the two terms of any given query into normal form.

Rewriting a term s amounts to finding a subterm $s|_p$ such that $s|_p = l\sigma$ for some rule $l \Rightarrow r$ and substitution σ. Then we say that l *matches* $s|_p$, and that $s|_p$ is an *instance* of l. If a TRS R is terminating, then it must be the case that $vars(l) \supseteq vars(r)$ for every rule $l \Rightarrow r$ in R, so that in each rewrite step the substitution σ is completely determined by matching.

Now we have seen that the two important properties that make TRS very useful for deduction are confluence and termination. Let us now see how they can be assured in practice.

1.3.1 How to ensure termination?

Termination of a TRS is undecidable, i.e., there is no algorithm that, when given as input a TRS R, tells us in finite time whether R is terminating or not. This is not surprising, since for every Turing machine there exists a (even one-rule) TRS that simulates it; in other words, rewriting is a Turing-complete computation mechanism. However, in spite of the fact that termination is undecidable, there exist general-purpose methods based on term orderings that frequently suffice in practice.

A (strict partial) *ordering* on $T(\mathcal{F},\mathcal{X})$ is an irreflexive transitive relation $\succ$. It is a *reduction* ordering if it is well-founded and monotonic, and *stable under substitutions*, that is, $s \succ t$ implies $s\sigma \succ t\sigma$ for all substitutions σ.

It turns out that a TRS R is terminating if, and only if, R is contained in a reduction ordering, that is, there exists some reduction ordering $\succ$ such that $l \succ r$ for every rule $l \Rightarrow r$ of R. The reader may prove this as an exercise (hint: show that if R terminates then $\Rightarrow_R{}^+$ itself is a reduction ordering).

But unconstrained term rewriting cannot deal properly with inherently non-terminating equations such as the commutativity axiom for a binary function symbol. Therefore the concepts have been generalized to ordered term rewriting. An *ordered* TRS is a pair $(E, \succ)$ where E is a set of equations and $\succ$ is a reduction ordering. Rewriting with an ordered TRS is done by applying equations of E in whatever direction agrees with $\succ$: the *ordered rewriting relation* defined by $(E, \succ)$ is the smallest monotonic binary relation $\Rightarrow_{E,\succ}$ on terms such that $s\sigma \Rightarrow_{E,\succ} t\sigma$ whenever $s \simeq t \in E$ (or $t \simeq s \in E$) and $s\sigma \succ t\sigma$. Hence an ordered TRS can handle *any* set of equations E, being terminating by definition. Note that $\Rightarrow_{E,\succ}$ coincides with $\Rightarrow_{E^\succ}$ where $E^\succ$ is the set of rewrite rules $s\sigma \Rightarrow t\sigma$ such that $s \simeq t \in E$ (or $t \simeq s \in E$), and σ a substitution such that $s\sigma \succ t\sigma$. Note also that since reduction orderings are only partial orderings on $T(\mathcal{F}, \mathcal{X})$, the relation $\Leftrightarrow_E^*$ is in general a strict superset of $\Leftrightarrow_{E^\succ}^*$. Except for trivial cases there are no reduction orderings that are total on $T(\mathcal{F}, \mathcal{X})$. Fortunately, reduction orderings that are total on ground terms do exist. For such a *complete reduction ordering* $\succ$, the relations $\Leftrightarrow_E^*$ and $\Leftrightarrow_{E^\succ}^*$ coincide on $T(\mathcal{F})$.

1.3.2 How to get reduction orderings in practice?

In practical deduction systems, useful reduction orderings can be obtained by easily automatisable, general-purpose methods. Here we will explain one of them, the so-called recursive path ordering (RPO). It is based on a well-founded ordering $\succ_\mathcal{F}$ on the function symbols $\mathcal{F}$.[2] Here $\succ_\mathcal{F}$ is called the *precedence*.

Let $\mathcal{F}$ be the disjoint union of two sets *lex* and *mul*, the symbols with *lexicographic* and *multiset status*, respectively, and let $=_{mul}$ denote the equality of terms up to the permutation of direct arguments of symbols with multiset status; for example, if $f \in mul$ and $g \in mul$, then $f(a, g(b, c, d), e) =_{mul} f(e, a, g(c, b, d))$.

Then RPO is defined as follows: $s \succ_{\text{rpo}} t$ if t is a variable that is a proper subterm of s or $s = f(s_1, \ldots, s_m)$ and $t = g(t_1, \ldots, t_n)$ and

1. $s_i \succ_{\text{rpo}} t$ or $s_i =_{mul} t$, for some i with $1 \leq i \leq m$ or
2. $f \succ_\mathcal{F} g$, and $s \succ_{\text{rpo}} t_j$, for all j with $1 \leq j \leq n$ or
3. $f = g$, $f \in lex$, $\langle s_1, \ldots, s_n\rangle \succ_{\text{rpos}}^{\text{lex}} \langle t_1, \ldots, t_n\rangle$, and $s \succ_{\text{rpo}} t_j$, for all j with $1 \leq j \leq n$
4. $f = g$, $f \in mul$, $\{s_1, \ldots, s_n\} \succ_{\text{rpos}}^{\text{mul}} \{t_1, \ldots, t_n\}$

where $\langle s_1, \ldots, s_n\rangle \succ_{\text{rpos}}^{\text{lex}} \langle t_1, \ldots, t_n\rangle$ if $\exists j \leq n$ s.t. $s_j \succ_{\text{rpo}} t_j$ and $\forall i < j$ $s_i =_{mul} t_i$. Furthermore, $\succ_{\text{rpos}}^{\text{mul}}$ is the multiset extension of $\succ_{\text{rpo}}$, defined as the smallest ordering such that for any $m \geq 0$, $S \cup \{s\} \succ_{\text{rpos}}^{\text{mul}} S' \cup \{t_1, \ldots, t_m\}$ whenever S is equal to S' up to $=_{mul}$ and $s \succ_{\text{rpo}} t_i$ for all i in $1 \ldots n$.

[2] This allows $\mathcal{F}$ to be infinite which is useful when infinitely many Skolem constants are needed.

The *lexicographic path ordering* is the particular case of RPO where $\mathcal{F} = lex$, and the *multiset path ordering* (or RPO without status) is the particular case where $\mathcal{F} = mul$.

For example, if $f \in mul$ and $g \in lex$, and $f \succ_{\mathcal{F}} g \succ_{\mathcal{F}} a \succ_{\mathcal{F}} b$, then $f(b, g(a, b, b), b) \succ_{\text{rpo}} g(a, f(a, g(b, a, a)), a)$. Exercise: explain why.

The RPO is a reduction ordering for every choice of mul, lex and $\succ_{\mathcal{F}}$. However in practice it is important that these be chosen appropriately so that deduction takes place with appropriate normal forms of terms and clauses.

1.3.3 How to ensure confluence?

Once we know that termination can be obtained by working with reduction orderings, we can apply Newman's lemma: terminating locally-confluent relations are confluent. Now let us analyze in which situations local confluence of (ordered) rewriting fails.

Assume a term t can be rewritten in two different ways, with instances by, respectively, substitutions σ and σ' of equations $l \simeq r$ and $l' \simeq r'$, at positions p and p', respectively. We may assume that variables in the two (not necessarily different) equations have been renamed so that the domains of σ and σ' are disjoint. There are three possible situations:

1. p and p' are disjoint positions (no one is prefix of the other). Then we have:
$$\begin{array}{ccc} & t[l\sigma]_p[l'\sigma']_{p'} & \\ \swarrow & & \searrow \\ t[r\sigma]_p[l'\sigma']_{p'} & & t[l\sigma]_p[r'\sigma']_{p'} \end{array}$$
But this does not lead to failure of local confluence, because $t[r\sigma]_p[l'\sigma']_{p'}$ and $t[l\sigma]_p[l'\sigma']_{p'}$ are *joinable*, that is, they can be rewritten into the same term (in this case, in one step):
$$\begin{array}{ccc} t[r\sigma]_p[l'\sigma']_{p'} & & t[l\sigma]_p[r'\sigma']_{p'} \\ \searrow & & \swarrow \\ & t[r\sigma]_p[r'\sigma']_{p'} & \end{array}$$
For example, assume $a \simeq b$ and $c \simeq d$ are equations and $f(a, c)$ rewrites into $f(b, c)$ and into $f(a, d)$. Then we have:
$$\begin{array}{ccc} & f(a,c) & \\ \swarrow & & \searrow \\ f(b,c) & & f(a,d) \\ \searrow & & \swarrow \\ & f(b,d) & \end{array}$$
2. If, say, p is a prefix of p', then $p' = p.q$ for some position q, and the situation is called an *overlap*. Then
$$\begin{array}{ccc} & t[l\sigma[l'\sigma']_q]_p & \\ \swarrow & & \searrow \\ t[r\sigma]_p & & t[l\sigma[r'\sigma']_q]_p \end{array}$$

and q is a position inside $l\sigma$.
(a) If some prefix of q is a variable x of l, then the overlap is called a *variable overlap* which also does not lead to a failure of local confluence. Let us see why both terms are joinable. The step with $l' \to r'$ rewrites $x\sigma$ into some term r''. Let σ' be extended to the variables in l and r such that $\sigma'(x) = r''$ and $\sigma'(y) = \sigma(y)$ if $y \neq x$. Assuming that x occurs n times in r and m times in l, we have:

$$\begin{tikzcd}
t[r\sigma]_p \arrow[ddrr, "n"] & & & & t[l\sigma[r'\sigma']_q]_p \arrow[dl, "m-1"] \\
& & & t[l\sigma']_p \arrow[dl] & \\
& & t[r\sigma']_p & &
\end{tikzcd}$$

For example, if $a \simeq b$ and $f(x,x,x,x) \simeq g(x,x)$ are applied to rewrite $f(a,a,a,a)$ into $g(a,a)$ and into $f(a,b,a,a)$, then we have:

$$\begin{tikzcd}
& & f(a,a,a,a) \arrow[dll] \arrow[drr] & & \\
g(a,a) \arrow[ddrr, "2"] & & & & f(a,b,a,a) \arrow[dl, "3"] \\
& & & f(b,b,b,b) \arrow[dl] & \\
& & g(b,b) & &
\end{tikzcd}$$

(b) If no prefix of q is a variable x of l, then $l|_q$ is a function symbol. This case is called a *critical overlap* and can be a situation of failure of local confluence. For example, if $f(a,x) \simeq g(x)$ and $h(f(y,b)) \simeq h(y,y)$ can be applied, according to $\succ$, as follows:

$$\begin{tikzcd}
& h(f(a,b)) \arrow[dl] \arrow[dr] & \\
h(g(b)) & & h(a,a)
\end{tikzcd}$$

then $h(g(b))$ and $h(a,a)$ may not be joinable.

From these consideration it follows that we only have to consider critical overlaps. How can we detect them? How can we check whether they witness a counterexample to local confluence? And what can we do if this is indeed the case?

1.3.4 How can we detect critical overlaps?

We have to detect all terms that contain overlapped *common instances* of left hand sides of possible rewrite steps. But this can be detected by *unification*: two terms s and t are *unifiable* if there exists some substitution σ such that $s\sigma = t\sigma$, and then σ is called a *unifier* of s and t. There exist unification algorithms that, when given terms s and t compute their *most general unifier* $mgu(s,t)$, that is, a unifier σ such that any other unifier θ of s and t is an instance of σ (there is some θ' such that $x\theta = x\sigma\theta'$ for all x).

In order to get a bit more intuition about unification, let us now briefly see what unification algorithms do (for more details, see the other courses in this summer school). The following rule set can be applied with different strategies leading to different algorithms. It works on unification problems that are sets of (unordered) pairs of terms $\{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$, for which a simultaneous mgu σ is computed for which $s_i\sigma = t_i\sigma$ for all $i \in 1\ldots n$:

$$
\begin{array}{lll}
0. & P \cup \{t \doteq t\} & \Rightarrow P \\
1. & P \cup \{f(t_1,\ldots,t_n) \doteq f(s_1,\ldots,s_n)\} & \Rightarrow P \cup \{t_1 \doteq s_1, \ldots, t_n \doteq s_n\} \\
2. & P \cup \{f(t_1,\ldots,t_n) \doteq g(s_1,\ldots,s_n)\} & \Rightarrow \text{fail}, \quad \text{if } f \neq g \\
3. & P \cup \{x \doteq t\} & \Rightarrow P\{x \mapsto t\} \cup \{x \doteq t\}, \\
 & & \quad \text{if } x \in vars(P),\ x \notin vars(t), \\
 & & \quad \text{and either } t \text{ is not a variable,} \\
 & & \quad \text{or else } t \in vars(P) \\
4. & P \cup \{x \doteq t\} & \Rightarrow \text{fail} \quad \text{if } x \in vars(t) \text{ and } x \neq t
\end{array}
$$

It is easy to see that the rules transform any unification problem into an equivalent one (having the same set of unifiers), with "fail" representing the empty set of unifiers. Any problem in normal form, that is, to which none of the rules can be applied anymore, is a problem of the form $\{x_1 \doteq t_1, \ldots, x_k \doteq t_k\}$ with x_i pairwise different and not occurring in any of the terms t_j. Such problems are trivially solved in that the substitution sending x_i to t_i is an mgu. Note that mgu's are unique up to renaming of variables in the following sense: if $x = y$ appears at some stage in the problem, then either all x's elsewhere in the problem are replaced by y's, and $x \mapsto y$ will be in the mgu, or the y's are replaced by x's, and $y \mapsto x$ is in the mgu. Also note that no new variables other than those occurring in initial unification problems are used to represent mgus.

Exercise: it remains to show that any strategy of applying the rules terminates. To do this, consider a well-founded ordering on unification problems comparing (i) their number of unsolved variables and (ii) (in case of the same number of unsolved variables), their number of function symbols.

Reconsidering the critical overlap produced by the rules $h(f(y,b)) \Rightarrow h(y,y)$ and $f(a,x) \Rightarrow g(x)$:

$$
\begin{array}{ccc}
 & h(f(a,b)) & \\
\swarrow & & \searrow \\
h(g(b)) & & h(a,a)
\end{array}
$$

we can see that it corresponds to the unification of $f(a,x)$, which is used as a left hand side, with the subterm $f(y,b)$ of the term used as left hand side of the other step.

It is also clear that, in general, if the rules are $l \Rightarrow r$ and $l' \Rightarrow r'$ respectively, then the two terms obtained from the critical overlap are always $l[r']\theta$ and $r\theta$, for some unifier θ of l' and a non-variable $l|_p$. Moreover, if $\sigma = mgu(l', l_p)$ then the two terms are always an instance of $l[r']\sigma$ and $r\sigma$. The equation $l[r']\sigma \simeq r\sigma$ is then called a *critical pair* between the rules are $l \Rightarrow r$ and $l' \Rightarrow r'$. It represents a real critical overlap only if it corresponds to applications of the two rules in this direction according to the current $\succ$, that is, if there exists some substitution θ such that $l\sigma\theta \succ r\sigma\theta$ and $l'\sigma\theta \succ r'\sigma\theta$.

Now such a a critical pair is nothing else than the conclusion of an inference by *superposition*, the following inference rule between equations, that is parameterized by the reduction ordering $\succ$:

$$\frac{l' \simeq r' \qquad l \simeq r}{(l[r']_p \simeq r)\sigma}$$

where $\sigma = mgu(l', l|_p)$, l_p is not a variable, and there exists some substitution θ such that $l\sigma\theta \succ r\sigma\theta$ and $l'\sigma\theta \succ r'\sigma\theta$. Note that an implicit assumption on all inference rules that we present in this paper is that variables are appropriately renamed so that the premises of an inference rule do not share any variables.

Now we can summarize the results of this subsection: if all critical pairs between the equations of E are joinable by ordered rewriting with $(E, \succ)$, then $(E, \succ)$ is convergent.

1.3.5 What can we do if some critical pair is not joinable?

To answer this question, note that by Birkhoff's theorem, every critical pair between equations of E is a logical consequence of E. Hence if we add the critical pair to E, we do not change its logical meaning. Moreover, it restores local confluence for its critical overlap.

Returning to our previous example, adding the critical pair $h(g(b)) \simeq h(a, a)$ solves the situation of non-local confluence

$$\begin{array}{ccc} & h(f(a,b)) & \\ \swarrow & & \searrow \\ h(g(b)) & & h(a,a) \end{array}$$

independently of whether $h(g(b)) \succ h(a, a)$ or $h(a, a) \succ h(g(b))$. On the other hand, to not render ordered rewriting incomplete, we have to require that each ground instance of each equation, be it an input equation or an equation generated by a critical pair inference, be orientable under the ordering. Therefore in the following we will make the usual assumption that $\succ$ is a complete reduction ordering, that is, is *total* on $T(\mathcal{F})$. RPO's with no function symbols of multiset status and with a total precedence are examples of complete reduction orderings. Being able to orient ground instances of equations is sufficient for our purposes. We want to solve word problems

$E \models s \simeq t$ by rewriting s and t to their normal forms, and hence may consider the variables of s and t to be (Skolem) constants. In other words, for solving word problems it suffices to rewrite ground terms (with ordered instances of possibly non-ground equations). Consequently, we may reformulate the notion of a critical pair as well as the superposition inference rule by requiring that there exists some *ground* substitution θ such that $l\sigma\theta \succ r\sigma\theta$ and $l'\sigma\theta \succ r'\sigma\theta$.

These ordering restrictions can be used in practice for reducing the number of critical pairs to be considered in several ways. By *ordering constraint* solving techniques, the restrictions can be decided for RPO and its variants. But since this can be expensive, instead of really checking whether there exists a θ such that, say, $l\sigma\theta \succ r\sigma\theta$, one can also use approximations. For instance, if $r\sigma \succ l\sigma$ (in a version of $\succ$ for terms with variables) then the critical pair is not needed.

Now, once we have added all non-joinable critical pairs, is the resulting E confluent? Unfortunately this is not the case (although this is not surprising, because otherwise it would provide a decision procedure for the word problem, which is undecidable in general). The reason is that the critical pairs that have been added can produce new critical pairs, which then have to be added, and so on. The (possibly non-terminating) procedure that does this until it obtains a set E all whose critical pairs are joinable, is called the (unfailing) Knuth-Bendix completion procedure.

Currently, most, if not all, state-of-the-art theorem provers for purely equational logic are variants of such Knuth-Bendix completion procedures. Indeed, they are semi-decision procedures for the word problem: if $E \models s \simeq t$, then completion with input E and a reduction ordering $\succ$ that is total on $T(\mathcal{F})$ will generate in finite time enough equations to rewrite s and t into the same normal form (provided it is applied with a *fair* control, i.e., no superposition inference is postponed forever). We will not prove this fact at this point, because it is a consequence of more general results that will be given later on.

1.3.6 First-order vs. inductive consequences

In many cases, the intended meaning of a specification E is not its standard first-order semantics, i.e., the class of all its models, but rather a more specific model I. Perhaps the best known example is the *initial* or *minimal Herbrand model* semantics, where I is the unique minimal Herbrand model of a set of Horn clauses with or without equality E (an algebraic specification, a logic program, a deductive data base, ...). Other interesting semantics that are used in practice as well include *final* semantics, or, for possibly non-Horn specifications E, *perfect model* semantics or the class of all (minimal) Herbrand models.

For example, let $\mathcal{F}$ be $\{0, s, +\}$ and assume E consists of the equations $0 + x \simeq x$ and $s(x) + y \simeq s(x + y)$. Then I is isomorphic to the algebra of natural numbers with $+$ and hence properties like the commutativity of $+$ are true in I. Exercise: prove $I \models x+y \simeq y+x$, and prove that $E \not\models x+y \simeq y+x$.

Properties true in I are called *inductive* theorems, since they usually require induction to prove them. However, there are so-called *implicit* induction methods that do not require the user of a (semi-)automatic inductive theorem prover to provide explicit induction schemes. Instead, induction can be done automatically, using well-founded orderings $\succ$ on terms, such as the ones we have seen earlier in this section.

Some implicit induction methods like the *proof-by-consistency* approach of [Bac88] are closely related to completion. Another simple one that is not based on completion, but rather on the notions of redundancy that we will see in the next section, is given in [Nie99]. A more detailed analysis of inductive proofs is unfortunately out of the scope of these notes, and in what follows we will only consider classical first-order semantics of equational theories.

1.4 Superposition for general clauses

The ideas on ordering restrictions explained so far for the case of unit equations can be extended to obtain refutationally complete procedures for general clauses. Then superposition becomes a restricted version of the following inference rule of *paramodulation* [RW69]:

$$\frac{C \vee s \simeq t \qquad D}{(\ C \vee D[t]_p\ )\sigma} \qquad \text{if } \sigma = mgu(s, D|_p)$$

In essence, superposition is paramodulation where $D|_p$ is not a variable, and where inferences take place *with* and *on maximal* (with respect to $\succ$) sides of *maximal* literals (in this paramodulation inference rule, we say that the inference takes place *with* s *on* $D|_p$).

In the following we further develop this idea, starting with the simple case of ground Horn clauses, and introducing the more general techniques as we go along.

1.4.1 Refutational completeness

An inference system $\mathcal{I}nf$ is called *refutationally complete* if from every inconsistent (unsatisfiable) set S of clauses the empty clause $\Box$ can be inferred by $\mathcal{I}nf$.

Here we will introduce the *model generation method*, one of the main methods for showing the refutational completeness of paramodulation-based inference systems, and the one that has been used for obtaining most results on deduction with symbolic constraints in this context. The main idea is as follows. Assume that S is *closed* under the given inference system $\mathcal{I}nf$ (that is, S contains all conclusions of inferences by $\mathcal{I}nf$ with premises in S) and assume $\Box \notin S$. Then it suffices to show that S is satisfiable. This is done by constructing an equality Herbrand model $\Leftrightarrow^*_{R_S}$ for S, where R_S is a ground TRS that is *generated* from S (remember that a congruence on $T(\mathcal{F})$ like $\Leftrightarrow^*_{R_S}$ is an equality Herbrand interpretation). In the following, instead of $\Leftrightarrow^*_R$, we will usually write R^*. Later on, the requirement of S to be closed with respect to the inference system $\mathcal{I}nf$ will be weakened into *saturatedness* with respect to $\mathcal{I}nf$, that is, closure up to *redundant* inferences.

1.4.2 The model generation method

We first consider the simple case of *ground Horn clauses*. In the following, let $\succ$ be a given total reduction ordering on ground terms. We will use the fact that such orderings also enjoy the *subterm property*, that is, $\succ \supseteq \rhd$, where $\rhd$ denotes the strict subterm ordering, defined: $s \rhd t$ if, and only if, $s|_p = t$ for some $p \neq \lambda$. Exercise: prove that any total reduction ordering on $T(\mathcal{F})$ is a simplification ordering, i.e., it has the subterm property.

Let $\Gamma \to l \simeq r$ be a ground clause where Γ is of the form $\{u_1 \simeq v_1, \ldots, u_n \simeq v_n\}$. The term l is called *strictly maximal* in $\Gamma \to l \simeq r$ if it is greater (with respect to $\succ$) than any other term in the clause, that is, if

$$l \succ r \ \wedge \ l \succ u_1 \ \wedge \ l \succ v_1 \ \wedge \ldots \wedge \ l \succ u_n \ \wedge \ l \succ v_n$$

In the following, this expression will be denoted by $l \succ r, \Gamma$. Similarly, l is called *maximal* in $\Gamma \to l \simeq r$ if $l \succeq t$, for each term t in r, Γ, where $\succeq$ denotes the reflexive closure of $\succ$. In the latter case we will also write $l \succeq r, \Gamma$. The inference system $\mathcal{G}$ for ground Horn clauses with equality is the following:

superposition right:

$$\frac{\Gamma' \to l \simeq r \qquad \Gamma \to s \simeq t}{\Gamma', \Gamma \to s[r]_p \simeq t} \quad \text{if} \quad \begin{array}{l} s|_p = l \text{ and} \\ l \succ r, \Gamma' \text{ and } s \succ t, \Gamma. \end{array}$$

superposition left:

$$\frac{\Gamma' \to l \simeq r \qquad \Gamma, s \simeq t \to \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \to \Delta} \quad \text{if} \quad \begin{array}{l} s|_p = l \text{ and} \\ l \succ r, \Gamma', \ s \succ t \text{ and } s \succeq \Gamma, \Delta. \end{array}$$

reflexivity resolution:

$$\frac{\Gamma, s \simeq s \to \Delta}{\Gamma \to \Delta} \quad \text{if} \quad s \succeq \Gamma, \Delta.$$

Given an ordering $\succ$ on a set Z, we define $\succ_{mul}$, the *multiset extension* of $\succ$, to be the smallest ordering on multisets over Z satisfying $M \cup \{s\} \succ_{mul} M \cup \{t_1, \ldots, t_n\}$ whenever $s \succ t_i$, for all $i \in 1 \ldots n$, with $n \geq 0$. In other words, for obtaining a smaller multiset one (repeatedly) replaces elements by finitely many, possibly zero, smaller elements. If $\succ$ is a well-founded and total ordering on Z, so is $\succ_{mul}$ on finite multisets over Z. This will be used to lift orderings $\succ$ on terms to orderings on equations and clauses. Let C be a ground clause, and let $emul(s \simeq t)$ be $\{s,t\}$ if $s \simeq t$ is a positive equation in C, and $\{s,s,t,t\}$ if it is negative. Then, if $\succ$ is an ordering, we define the ordering $\succ_e$ on (occurrences of) ground equations in a clause by $e \succ_e e'$ if $emul(e) \succ_{mul} emul(e')$. Similarly, $\succ_c$ on ground clauses is defined $C \succ_c D$ if $mse(C)\ \ (\succ_{mul})_{mul}\ \ mse(D)$, where $mse(C)$ is the multiset of all $emul(e)$ for occurrences e of equations in C. We see that $C \succ_c D$ whenever (i) the maximal term in C is strictly larger then the maximal term in D, or (ii) the maximal term in C and D is the same, and it occurs negatively in C but only positively in D, or (iii) D results from C by replacing an occurrence of a subterm in some literal by a smaller term, or deleting a literal, and adding any (finite, including zero) number of smaller literals. These are three crucial properties[3] of $\succ_c$ which have the effect that the inferences in $\mathcal{G}$ are *monotone* in that the conclusion is smaller than the maximal (the second, in the case of superposition) premise. In fact, reflexivity resolution deletes a literal. Superposition replaces the maximal term in the second premise by a smaller term and appends side literals (in Γ') which are all smaller than the literal $s \simeq t$ in which the replacement occurs.

Definition 1. Let S be a set of ground Horn clauses. We now define, by induction on $\succ_c$, the ground TRS R_S *generated* by S, as follows. A clause C in S of the form $\Gamma \rightarrow l \simeq r$ *generates* the rule $l \Rightarrow r$ if

1. $R_C^* \not\models C$,
2. $l \succ r, \Gamma$, and
3. l is irreducible by R_C

where R_C is the set of rules generated by all clauses D in S such that $C \succ_c D$. We denote by R_S (or simply R when S is clear from the context) the set of rules generated by all clauses of S.

Let us analyze the three conditions. The first one states that a clause only contributes to the model if it does not hold in the partial model built so far and hence we are *forced* to extend this partial model. The second one states that a clause can only generate a rule $l \Rightarrow r$ if l is the strictly maximal term of the clause, and hence $l \succ r$, which makes R a terminating TRS. The third condition, stating that l is irreducible by the rules generated so far, is, together with the second one, the key for showing that R is convergent, which is in turn essential for the completeness result.

Lemma 1. *For every set of ground clauses S, the set of rules R generated for S is convergent (i.e., confluent and terminating). Furthermore, if $R_C^* \models C$ then $R^* \models C$, for all C in S.*

[3] The clause ordering also satisfies certain other properties in addition to (i)—(iii). These are not essential. In fact, to obtain completeness results of certain variants of superposition, different clause orderings, but also satisfying (i)–(iii), are required.

Proof. On the one hand, R terminates since $l \succ r$ for all its rules $l \Rightarrow r$. On the other hand, it suffices to show *local* confluence, which in the ground case is well-known (and easily shown) to hold if there are no two different rules $l \Rightarrow r$ and $l' \Rightarrow r'$ where l' is a subterm of l. This property is fulfilled: clearly when a clause C generates $l \Rightarrow r$, no such $l' \Rightarrow r'$ is in R_C; but if $l' \Rightarrow r'$ is generated by a clause D with $D \succ_c C$ then, by definition of $\succ_c$, we must have $l' \succ l$ and hence l' cannot a be subterm of l either.

To show $R_C^* \models C$ implies $R^* \models C$, let C be $\Gamma \to \Delta$, and assume $R_C^* \models C$. If $R_C^* \models \Delta$ then $R^* \models \Delta$ since $R \supseteq R_C$. Otherwise, $R_C^* \not\models \Gamma$. Then $R^* \not\models \Gamma$ follows from the fact that if a term t occurs negatively in a clause that clause is larger than any clause in which t is also maximal but occurs only positively. As all rules in $R \setminus R_C$ are generated by clauses bigger than C with their maximal term occurring only positively, these clauses generate rules that have left hand sides too big to reduce any term occurring in Γ. Since R is convergent this implies $R^* \not\models \Gamma$. $\square$

Theorem 1. *The inference system $\mathcal{G}$ is refutationally complete for ground Horn clauses.*

Proof. Let S be a set of ground Horn clauses that is closed under $\mathcal{G}$ and such that $\square \notin S$. We prove that then S is satisfiable by showing that R^* is a model for S. We proceed by induction on $\succ_c$, that is, we derive a contradiction from the existence of a minimal (with respect to $\succ_c$) clause C in S such that $R^* \not\models C$. Clauses false in R^* are called *counterexamples* for the *candidate model* R. Therefore we assume that C is a minimal counterexample (and different from the empty clause). There are a number of cases to be considered, depending on the occurrences in C of its maximal term s, i.e., the term s such that $s \succeq u$ for all terms u in C (s is unique since $\succ$ is total on $T(\mathcal{F})$):

1. s occurs only in the succedent and C is $\Gamma \to s \simeq s$. This is not possible since $R^* \not\models C$.
2. s occurs only in the succedent and C is $\Gamma \to s \simeq t$ with $s \neq t$. Then $s \succ t, \Gamma$. Since $R^* \not\models C$, we have $R^* \supseteq \Gamma$ and $s \simeq t \notin R^*$, i.e., C has not generated the rule $s \Rightarrow t$. From Lemma 1 we infer that $R_C^* \not\models C$ so that the reason why C has not generated a rule must be that s is reducible by some rule $l \Rightarrow r \in R_C$. Suppose that $l \Rightarrow r$ has been generated by a clause C' of the form $\Gamma' \to l \simeq r$. Then there exists an inference by superposition right:

$$\frac{\Gamma' \to l \simeq r \qquad \Gamma \to s \simeq t}{\Gamma', \Gamma \to s[r]_p \simeq t}$$

where the conclusion D has only terms u with $s \succ u$, and hence $C \succ_c D$. Moreover, D is in S and $R^* \not\models D$, since $R^* \supseteq \Gamma \cup \Gamma'$ and $s[r]_p \simeq t \notin R^*$ (since otherwise $s[l]_p \simeq t \in R^*$). This contradicts the minimality of C.

3. s occurs in the antecedent and C is $\Gamma, s \simeq s \rightarrow \Delta$. Then there exists an inference by reflexivity resolution:

$$\frac{\Gamma, s \simeq s \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

and for its conclusion D we have $C \succ_c D$. Moreover, D is in S and $R^* \not\models D$, which is a contradiction as in the previous case.
4. s occurs in the antecedent and C is $\Gamma, s \simeq t \rightarrow \Delta$ with $s \succ t$. Since $R^* \not\models C$, we have $s \simeq t \in R^*$ and since R is convergent, s and t must have the same normal forms with respect to R, so s must be reducible by some rule $l \Rightarrow r \in R$. Assume $l \Rightarrow r$ has been generated by a clause C' of the form $\Gamma' \rightarrow l \simeq r$. Then there exists an inference by superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \qquad \Gamma, s[l]_p \simeq t \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta}$$

and for its conclusion D we have that $C \succ_c D$. Moreover, D is in S and $R^* \not\models D$, which again contradicts the minimality of C. □

Example 1. In the figure 1 we illustrate how the rewrite system R changes during the process of saturating a set of ground clauses under $\mathcal{G}$. None of the intermediate interpretations is a model. In the end we obtain one, as stated by the theorem.

Consider the lexicographic path ordering generated by the precedence $f > a > b > c > d$. The following table shows in the left column the ground Horn clauses (sorted in ascending order) at each step of the closure process. The first set is the initial clause set, the two other sets are obtained by adding an inference with the minimal counterexample of the preceding set of clauses. In the right column the rewrite systems as generated by the respective clause sets are displayed. The maximal term of every clause is underlined and the subterms of the clauses involved in the inference leading to the next clause set are framed.

1.4.3 Selection of negative equations

The inference system $\mathcal{G}$ includes strong ordering restrictions: a superposition inference is needed only if the term involved is maximal in the clause, and even strictly maximal for terms of positive equations. But more constraints can be imposed. If a clause C has a non-empty antecedent, it is possible to arbitrarily *select* exactly one of its negative equations. Then completeness is preserved if C is not used as left premise of any superposition inference and the only inferences involving C are equality resolution or superposition left on its selected equation.

S	R
$\rightarrow \boxed{\underline{c}} \simeq d$ $\underline{f(d)} \simeq d \rightarrow a \simeq b$ $\rightarrow \underline{f(\boxed{c})} \simeq d$	$c \simeq d$
$\rightarrow \underline{c} \simeq d$ $\rightarrow \boxed{\underline{f(d)}} \simeq d$ $\boxed{\underline{f(d)}} \simeq d \rightarrow a \simeq b$ $\rightarrow \underline{f(c)} \simeq d$	$c \simeq d$ $f(d) \simeq d$
$\rightarrow \underline{c} \simeq d$ $d \simeq d \rightarrow \underline{a} \simeq b$ $\rightarrow \underline{f(d)} \simeq d$ $\underline{f(d)} \simeq d \rightarrow a \simeq b$ $\rightarrow \underline{f(c)} \simeq d$	$c \simeq d$ $a \simeq b$ $f(d) \simeq d$

Fig. 1.1. Evolution of the rewrite system during saturation

In the following variant of the inference system $\mathcal{G}$, selected equations are underlined, and it is assumed that any clause $\Gamma \rightarrow \Delta$ where the maximal term occurs in Γ has one selected negative equation.

superposition right:

$$\frac{\Gamma' \rightarrow l \simeq r \qquad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t} \quad \text{if} \quad \begin{array}{l} s|_p = l,\ l \succ r, \Gamma',\ s \succ t, \Gamma, \text{ and} \\ \text{no equation is selected in } \Gamma \text{ or } \Gamma'. \end{array}$$

superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \qquad \Gamma, \underline{s \simeq t} \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta} \quad \text{if} \quad \begin{array}{l} s|_p = l,\ l \succ r, \Gamma',\ s \succ t, \text{ and} \\ \text{no equation is selected in } \Gamma'. \end{array}$$

reflexivity resolution:

$$\frac{\Gamma, \underline{s \simeq s} \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

Exercise: adapt the proof of completeness of Theorem 1 to this framework with selection (hint: consider that clauses with selected equations generate no rules).

1.4.4 Clauses with variables

Up to now, in this section we have only dealt with ground clauses. Since a non-ground clause represents the set of all its ground instances, a refutationally complete method for the non-ground case would be to systematically enumerate all ground instances of the clauses, and to perform inferences by $\mathcal{G}$ between those instances. But fortunately it is possible to perform inferences between non-ground clauses, covering in one step a possibly large number of ground inferences. We now adapt $\mathcal{G}$ according to this view. For example, at the ground level, in the superposition right inference

$$\frac{\Gamma' \rightarrow l \simeq r \qquad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t} \quad \text{if} \quad \begin{array}{l} s|_p = l \text{ and} \\ l \succ r, \Gamma' \text{ and } s \succ t, \Gamma. \end{array}$$

we have required $s|_p$ and l to be the same term. At the non-ground level, this becomes a constraint $s|_p \doteq l$ on the possible instances of the conclusion. Therefore we may view the conclusion of a non-ground inference as a constrained clause $D \ \mid \ T$, representing, semantically, the set of ground instances $D\sigma$ for which the correspondingly instantiated constraint $T\sigma$ is true.

Hence if the conclusion is of the form $D \ \mid \ s|_p \doteq l \wedge \gamma$, the instances $D\sigma$ of D for which $s|_p\sigma \neq l\sigma$ are excluded, and similarly for ordering restrictions. For instance, instead of requiring $l \succ r$ as a meta-level condition of the inference, it becomes part of the constraint of the conclusion, excluding those instances of the conclusion that correspond to ground inferences between instances of the premises for which $l \succ r$ does not hold:

$$\frac{\Gamma' \rightarrow l \simeq r \qquad\qquad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t \quad \mid \ s|_p \doteq l \ \wedge \ l > r, \Gamma' \ \wedge \ s > t, \Gamma}$$

Therefore in our context, a *constraint* is a boolean combination of atomic constraints of the form $s \doteq t$ (an *equality constraint*) or $s > t$ or $s \geq t$ (*inequality constraints*). A substitution σ is called a *solution* of a constraint T if the instantiated constraint $T\sigma$ is ground and evaluates to true when interpreting $\doteq$ and $>$ respectively, as the syntactic equality and $\succ$ on ground terms. We shall also write $\sigma \models T$ whenever σ is a solution of T. Constraints such as $l > r, \Gamma$ are considered to be abbreviations for an obvious conjunction of inequality constraints involving the terms in Γ, l and r. A ground clause $D\sigma$ is called a *ground instance* of a constrained clause $D \ \mid \ T$ whenever σ is a solution of T. If C is the empty clause and T is satisfiable, the empty (unconstrained) clause is an instance of C. If T is unsatisfiable, a clause constrained by T does not have any instances and, hence, can be considered a tautology.

As in the case of the critical pair inference for unit equations, the inference rule may be equipped with the additional restriction excluding the cases where $s|_p$ is a variable. This indicates that certain inferences between ground instances of the premises are redundant, namely those for which the replacement of equals by equals occurs *inside* the substitution part. Such inferences are also called inferences *below variables*), as they occur at positions $s\sigma|_p$ where $s|_{p'}$ is a variable for some prefix p' of p.

The full inference system for non-ground Horn clauses will be called $\mathcal{H}$ (for Horn), where we again assume that variables in premises are renamed to achieve variable disjointness:

superposition right:

$$\frac{\Gamma' \rightarrow l \simeq r \qquad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[r]_p \simeq t \ \mid\ s|_p \doteq l \ \wedge\ l > r, \Gamma' \ \wedge\ s > t, \Gamma}$$ if $s|_p$ is not a variable, and nothing is selected in Γ', Γ

superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \qquad \Gamma, \underline{s \simeq t} \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta \ \mid\ s|_p \doteq l \ \wedge\ l > r, \Gamma' \ \wedge\ s > t}$$ if $s|_p$ is not a variable, and nothing is selected in Γ'

reflexivity resolution:

$$\frac{\Gamma, \underline{s \simeq t} \rightarrow \Delta}{\Gamma \rightarrow \Delta \quad \mid\ s \doteq t}$$

Example 2. Consider the lexicographic path ordering generated by the precedence $h > a > f > g > b$ and the inference

$$\frac{g(x) \simeq x \rightarrow f(a,x) \simeq f(x,x) \ \rightarrow h(f(a,g(y))) \simeq h(y)}{\begin{array}{l} g(x) \simeq x \rightarrow h(f(x,x)) \simeq h(y) \ \mid\ f(a,x) \doteq f(a,g(y)) \ \wedge\ h(f(a,g(y))) > h(y) \ \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad f(a,x) > f(x,x), g(x), x \end{array}}$$

The constraint of the conclusion is satisfiable: using the properties of the ordering and solving the unification problem, the constraint can be simplified to

$$x \doteq g(y) \quad \wedge \quad a > x$$

which is satisfied, for instance, by the substitution $\{y \mapsto b, x \mapsto g(b)\}$.

On the other hand, the following inference is not needed

$$\frac{\rightarrow f(x,x) \simeq f(a,x) \qquad \rightarrow f(g(y),z) \simeq h(z)}{\begin{array}{l} \rightarrow f(a,x) \simeq h(z) \quad \mid\ f(x,x) \doteq f(g(y),z) \ \wedge\ f(g(y),z) > h(z) \ \wedge \\ \qquad\qquad\qquad\qquad\quad f(x,x) > f(a,x) \end{array}}$$

since the constraint of its conclusion is unsatisfiable: it can be simplified to

$$x \doteq g(y) \quad \wedge \quad x \doteq z \quad \wedge \quad y \geq h(z) \quad \wedge \quad x > a$$

which implies $y \geq h(g(y))$ and therefore it is unsatisfiable. Note that if the equality constraint and the ordering constraint are considered independently then both are satisfiable.

1.4.5 Completeness without constraint inheritance

There are several possible treatments for constrained clauses as generated by the inference system $\mathcal{H}$. The classical view is to make them unconstrained again. That is, clauses of the form $C \;\mid\; s \doteq t \wedge OC$, for some ordering constraint OC, are immediately converted into $C\sigma$ where $\sigma = mgu(s,t)$. This will be called the strategy *without constraint inheritance*. Below we will also explain an alternative where constraints are kept and inherited throughout inferences.

Of course, the clause $C\sigma$ has to be generated only if the constraint $s \doteq t \wedge OC$ is satisfiable in $T(\mathcal{F})$. If $\succ$ is the lexicographic path ordering (LPO) the satisfiability of constraints is decidable [Com90,Nie93]. But traditionally in the literature weaker approximations by non-global tests are used; for example, inference systems are sometimes expressed with local conditions of the form $r \not\succeq l$ when in our framework we have $l > r$ as a part of the global constraint OC. Note that such weaker approximations do not lead to unsoundness, but only to the generation of unnecessary clauses.

In the following, we call a set of (unconstrained) Horn clauses S *closed under $\mathcal{H}$ without constraint inheritance* if $D\sigma \in S$ for all inferences by $\mathcal{H}$ with premises in S and conclusion $D \;\mid\; s \doteq t \wedge OC$ such that $s \doteq t \wedge OC$ is satisfiable and $\sigma = mgu(s,t)$.

Theorem 2. *The inference system $\mathcal{H}$ without constraint inheritance is refutationally complete for Horn clauses.*

Proof. Let S be a set of Horn clauses closed under $\mathcal{H}$ without constraint inheritance such that $\Box \notin S$. The proof is very similar to the one for $\mathcal{G}$: we exhibit a model R^* for S. We proceed again by induction on $\succ_c$, but now the rôle of the ground clauses in the proof for $\mathcal{G}$ is played by all *ground instances* of clauses in S, and the generation of rules in R from these ground instances is as for $\mathcal{G}$. Now we derive a contradiction from the existence of a minimal (with respect to $\succ_c$) *ground instance $C\sigma$ of a* clause C in S such that $R^* \not\models C\sigma$. The cases considered are the same ones as well, again depending on the occurrences in $C\sigma$ of its maximal term $s\sigma$.

The only additional complication lies in the *lifting* argument. As all the cases are essentially similar, we will only treat one of them in detail. Suppose that $C = \Gamma, \underline{s \simeq t} \rightarrow \Delta$ and $s\sigma \succ t\sigma$. Since $R^* \not\models C\sigma$, we have $s\sigma \simeq t\sigma \in R^*$ and since R is convergent, $s\sigma$ must be reducible by some rule $l\sigma \Rightarrow r\sigma \in R$, generated by a clause C' of the form $\Gamma' \rightarrow l \simeq r$. Now we have $s\sigma|_p = l\sigma$, and there are two possibilities:

An inference. $s|_p$ is a non-variable position of s.
Then there exists an inference by superposition left:

$$\frac{\Gamma' \rightarrow l \simeq r \qquad \Gamma, \underline{s \simeq t} \rightarrow \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \rightarrow \Delta \quad \mid \; s|_p \doteq l \;\wedge\; l > r, \Gamma' \;\wedge\; s > t}$$

where the conclusion $D \mid T$ has an instance $D\sigma$ (i.e., σ satisfies the constraint T) such that $C\sigma \succ_c D\sigma$, and $R^* \not\models D\sigma$, contradicting the minimality of $C\sigma$.

Lifting. $s|_{p'}$ is a variable x for some prefix p' of p.
Then $p = p' \cdot p''$ and $x\sigma|_{p''}$ is $l\sigma$. Now let σ' be the ground substitution with the same domain as σ but where $x\sigma' = x\sigma[r\sigma]_{p''}$ and $y\sigma' = y\sigma$ for all other variables y. Then $R^* \not\models C\sigma'$ and $C\sigma \succ_c C\sigma'$, contradicting the minimality of $C\sigma$. □Notice that the lifting argument relies on the existence of the substitution-reduced ground instance $C\sigma'$ of C. In inference systems without hereditary constraints, clauses in saturated clause sets carry no constraint. Therefore any grounding substitution yields an instance of C. For clauses with constraints the situation can be different. Reducing a solution σ of a constraint by a rewrite system might yield a substitution that is not a solution anymore, so that $C\sigma'$ is not a ground instance of C that could be a counterexample.

1.4.6 General clauses

Now we consider clauses that may have more than one equation in their succedent. For this we need the notions of *maximal* and *strictly maximal* equation in a clause C. To express maximality and strict maximality we use $gr(s \simeq t, \Delta)$ and $geq(s \simeq t, \Delta)$, which denote, respectively, that the equation $s \simeq t$, considered as a multiset of terms, is strictly greater (with respect to the multiset extension of $\succ$) than and greater than or equal to all equations in a multiset of equations Δ.

The inference system $\mathcal{I}$ for general clauses consists of these inference rules:

superposition right:

$$\frac{\Gamma' \to l \simeq r, \Delta' \qquad \Gamma \to s \simeq t, \Delta}{\Gamma', \Gamma \to s[r]_p \simeq t, \Delta', \Delta \quad \begin{array}{l} \mid\ s|_p \doteq l \ \wedge\ l > r, \Gamma' \ \wedge\ s > t, \Gamma \ \wedge \\ \ \ gr(l \simeq r, \Delta') \ \wedge\ gr(s \simeq t, \Delta) \end{array}}$$

superposition left:

$$\frac{\Gamma' \to l \simeq r, \Delta' \qquad \Gamma, \underline{s \simeq t} \to \Delta}{\Gamma', \Gamma, s[r]_p \simeq t \to \Delta', \Delta \quad \begin{array}{l} \mid\ s|_p \doteq l \ \wedge\ l > r, \Gamma' \ \wedge\ s > t \ \wedge \\ \ \ gr(l \simeq r, \Delta') \end{array}}$$

reflexivity resolution:

$$\frac{\Gamma, \underline{s \simeq t} \to \Delta}{\Gamma \to \Delta \quad \mid\ s \doteq t}$$

equality factoring:

$$\frac{\Gamma \to s \simeq t, s' \simeq t', \Delta}{\Gamma, t \simeq t' \to s \simeq t', \Delta \ \mid\ s \doteq s' \ \wedge\ s > t, \Gamma \ \wedge\ geq(s \simeq t, (s' \simeq t', \Delta))}$$

where as in the Horn case in both superposition rules $s|_p$ is not a variable and where premises do not have selected negative equations except the ones that are explicitly indicated by the inferences.

Example 3. Consider the lexicographic path ordering generated by the precedence $f > g > h$ and the following inference by superposition right

$$\frac{\rightarrow g(z) \simeq h(z) \qquad\qquad \rightarrow f(g(x),y) \simeq g(x), f(g(x),y) \simeq y}{\begin{array}{ll}\rightarrow f(h(z),y) \simeq g(x), f(g(x),y) \simeq y \ \mid & g(x) \doteq g(z) \ \wedge\ g(z) > h(z) \ \wedge \\ & f(g(x),y) > g(x) \ \wedge \\ & gr(f(g(x),y) \simeq g(x), \{f(g(x),y) \simeq y\})\end{array}}$$

where $gr(f(g(x),y) \simeq g(x), \{f(g(x),y) \simeq y\})$ represents the ordering constraint $\{f(g(x),y), g(x)\} >_{mul} \{f(g(x),y), y\}$, which can be simplified into $g(x) > y$. Now, simplifying the rest of the constraint the conclusion of the inference can be written as

$$\rightarrow f(h(z),y) \simeq g(x), f(g(x),y) \simeq y \ \mid \ x \doteq z \ \wedge\ g(x) > y$$

Theorem 3. *The inference system $\mathcal{I}$ is refutationally complete for general clauses.*

The proof is an extension of the proof of Theorem 2. One way to do it is to extend the construction of candidate models in Definition 1 appropriately. Let a ground clause $C = \Gamma \rightarrow l \simeq r, \Delta$ generate $l \simeq r$ if, and only if, (i) C is false in the partial interpretation R_C^*, if $l \succ r, \Gamma$, if $geq(l \simeq r, \Delta)$ holds, if l is irreducible by R_C^*, and (ii) no equation in Δ is true in $(R_C \cup \{l \simeq r\})^*$. Then the reasoning about reducing counterexamples is essentially similar. Because of condition (ii) preventing the production of certain rules, an additional form of counterexamples—those satisfying (i) but not (ii)—may occur. Counterexamples of this form can be reduced by equality factoring inferences.

Exercise: Prove the theorem on the basis of these hints.

A consequence of this theorem is compactness. In fact, the theorem makes no assumptions about the cardinality of S. If S is infinite and unsatisfiable, we may consider its closure $S_\infty = \bigcup_{i\geq 1} S^i$ under $\mathcal{I}$, with $S^0 = S$, and $S^{i+1} = S^i \cup \mathcal{I}(S)$, for $i \geq 0$. Then S_∞ is also unsatisfiable hence, by the theorem, contains the empty clause, which must then be an element in one of the S^i. Therefore, the empty clause has a finite superposition proof involving only finitely many clauses from S as initial premises.

1.4.7 Ordered Resolution with Selection

As pointed out in the section 1.2.7, one may view non-equational atoms of the form $p(s_1, \ldots, s_n)$, with p a predicate symbol, as equations of the form $p(s_1, \ldots, s_n) \simeq \top$. Ordered resolution is a specialization of superposition to *non-equational clauses*, that is, clauses in which all equations are encodings of non-equational atoms. In that case, superposition inferences are only possible at the top of terms (that encode an atom). Also, superposition, right, generates a positive equation $\top \simeq \top$ so that its conclusion is a tautology, hence redundant.[4] Similarly, superposition, left, produces a negative equation $\top \simeq \top$ which, when selected, can be eliminated by reflexivity resolution. Therefore reflexivity resolution inferences will be built into the resolution inference. Equality factoring specializes to factoring. Moreover, ordering constraints can be greatly simplified. In particular a constraint such as $gr(p(s_1, \ldots, s_n) \simeq \top, \Delta)$ simplifies to $p(s_1, \ldots, s_n) \succ \Delta$, assuming, as we do, that $\perp$ is minimal in the ordering. Also, $geq(p(s_1, \ldots, s_n) \simeq \top, \Delta)$ is true if and only if $p(s_1, \ldots, s_n)$ is the maximal atom with respect to Δ, justifying the simplified notation $geq(p(s_1, \ldots, s_n), \Delta)$.

The inference system $\mathcal{R}$ of ordered resolution with selection for non-equational clauses consists of these inference rules, where letters A and B denote non-equational atoms:

resolution:

$$\frac{\Gamma' \rightarrow A, \Delta' \qquad \Gamma, \underline{B} \rightarrow \Delta}{\Gamma', \Gamma \rightarrow \Delta', \Delta \qquad \mid \; A \doteq B \;\wedge\; A > \Gamma', \Delta'}$$

factoring:

$$\frac{\Gamma \rightarrow A, B, \Delta}{\Gamma \rightarrow A, \Delta \quad \mid \; A \doteq B \;\wedge\; A > \Gamma \;\wedge\; geq(A, \Delta)}$$

where, as in the case of $\mathcal{I}$, premises do not have selected negative atoms except for the ones that are explicitly indicated in the inferences.

From what we have said before, the refutational completeness of ordered resolution with selection is a special case of Theorem 3.

Theorem 4. *The inference system $\mathcal{R}$ is refutationally complete for general non-equational clauses.*

Note that since in resolution there is no rewriting of proper subterms, the requirements about the ordering can be relaxed. All that is needed is that the ordering is total and well-founded[5] on ground atoms $p(s_1, \ldots, s_n)$. The non-atomic terms s_i need not be ordered.

[4] Tautologies can neither be productive nor minimal counterexamples. Therefore they do not participate in essential inferences. For a general notion of redundancy where tautologies appear as a special case we refer to the next section.

[5] Even well-foundedness is not essential as, because of compactness, only finite sets of ground atoms participate in a refutation.

1.5 Saturation procedures

Simplification and deletion techniques that are compatible with refutational completeness are well-known to be essential for obtaining useful implementations of paramodulation-based theorem proving. These techniques can be uniformly covered by notions of *redundancy* for inferences and clauses. On the one hand, instead of closing the initial set of clauses under a given inference system, we only need to *saturate* it, that is, close it up to *redundant inferences*. Similarly, *redundant clauses* are unnecessary as well, and can be removed during the processes that compute saturated sets, called *saturation* procedures. In this setting, roughly speaking, a clause is redundant if all its instances are logical consequences of other clauses that are smaller. An inference is redundant if, for all its instances, either one of its premises is a strictly redundant clause or else its conclusion follows from clauses smaller than the maximal premise.

Here we only describe in detail the computation of saturated sets for the case of ground clauses. Hence in the remainder of this section all clauses, denoted by $C, D, \ldots$, and sets of clauses, denoted by S, are assumed to be ground. The different notions explained here smoothly extend to the non-ground case (see [BG98,NR01]).

Let S be a set of ground clauses. We denote by $S^{\prec C}$ the set of all D in S such that $C \succ_c D$. A clause C is called *redundant* (in S) if $S^{\prec C} \models C$. An inference with premises $C_1, \ldots, C_n$, maximal premise C and conclusion D is *redundant* (in S) if either one of its premises is redundant in S, or $S^{\prec C} \models D$. S is *saturated* with respect to an inference system $\mathcal{I}nf$ if every inference of $\mathcal{I}nf$ from premises in S is redundant in S.

Theorem 5. *Let S be a set of ground clauses that is saturated with respect to $\mathcal{I}$. Then $R^* \models S$ if, and only if, $\Box \notin S$, and hence S is unsatisfiable if, and only if, $\Box \in S$.*

For the proof of this theorem, one inspects the proof of Theorem 1 to find that inferences essential for reducing counterexamples to the candidate model R for a saturated set S of ground clauses involve rather specific kinds of premises. The maximal premise of such an inference is the (hypothetical) minimal counterexample D, and the side premise (if any) is a productive clause C. The minimal counterexample cannot be implied by smaller (hence true in R^*) clauses in S. For similar reasons, productive clauses C smaller than the minimal counterexample cannot be redundant either. (Otherwise they would already be true in the partial interpretation R_C^*.) Also, as the inference is monotone, the conclusion is a clause smaller than the maximal premise. Therefore, if the inference is redundant the conclusion is a counterexample smaller than D, but is also implied by clauses smaller than D, which together is impossible. Hence if S is saturated up to redundancy, R is a model of S, or else the minimal counterexample D in S must be the empty clause.

Exercise: Fill in the details of this proof sketch.

The previous theorem states that, instead of computing sets closed under the inference system, it suffices to saturate them. Therefore, we may now describe methods for computing saturated sets which are more practical than naive procedures which simply compute all inferences from existing clauses. These methods are abstractly formalized by the notion of a (theorem proving) *derivation* which are sequences of sets of clauses where each set is obtained by either adding some logical consequence to, or by removing some *redundant* clause from, the previous set in the sequence.

A *derivation* is a (possibly infinite) sequence of sets of clauses $S_0, S_1, \ldots$ where each S_{i+1} is either $S_i \cup S$, for some set of clauses S such that $S_i \models S$, or else S_{i+1} is $S_i \setminus S$, for some set of clauses S which are all redundant in S_i. Clauses belonging, from some k on, to all S_i with $i > k$ are called *persistent:* the set $S_\infty = \cup_k \cap_{i>k} S_i$ is the set of all persistent clauses.

A large number of different practical redundancy elimination methods that are applied in theorem provers fit into the notion of derivation. For example, simplifying $P(f(a))$ into $P(a)$ with the equation $f(a) \simeq a$, is modeled by first adding $P(a)$, and then removing $P(f(a))$ which has become redundant in the presence of $P(a)$ and $f(a) \simeq a$.

However, if our aim is to obtain refutationally complete theorem proving procedures by computing derivations, and in the limit, to obtain a saturated set, then a notion of *fairness* is required. A derivation $S_0, S_1, \ldots$ is *fair* with respect to an inference system $\mathcal{I}nf$ if every inference of $\mathcal{I}nf$ with premises in S_∞ is redundant in S_j for some j.

Now what can be done in practice to ensure fairness? On the one hand, for the inference system $\mathcal{I}$, if an inference is not redundant, it can *be made* redundant by adding its conclusion, since in $\mathcal{I}$ the conclusion always is smaller than the maximal premise. On the other hand, this has to be done only for inferences with persistent premises. Since it is not possible to know, at a certain point S_i, whether a given premise is going to be persisent or not, some means to ensure that no non-redundant inference is eternally postponed must be provided. In practice this is usually done by means of size-based priority queues.

Theorem 6. *If $S_0, S_1, \ldots$ is a fair derivation with respect to $\mathcal{I}nf$, then $\cup_i S_i$ is saturated with respect to $\mathcal{I}nf$, and hence if $\mathcal{I}nf$ is $\mathcal{I}$, then S_0 is unsatisfiable if, and only if, $\Box \in S_j$ for some j. Furthermore, if $S_0, S_1, \ldots, S_n$ is a fair derivation then S_n is saturated and logically equivalent to S_0.*

For the proof of this theorem it is essential to observe that redundancy of a clause or inference is preserved under adding clauses and removing redundant clauses.

1.5.1 Computing with saturated sets

In practice, indeed it is sometimes possible to obtain a finite saturated set S_n (not containing the empty clause) for a given input S_0. In this case hence its satisfiability has been proved. This kind of consistency proving has of course many applications and is also closely related to inductive theorem proving.

Theorem proving in theories expressed by saturated sets S of axioms is also interesting because more efficient proof strategies become (refutationally) complete. For example, if S and S' are two sets of clauses where S is saturated, then, when saturating $S \cup S'$, one can exploit the fact that inferences between premises of S are redundant. Note that such a *set-of-support* strategy is incomplete in general for ordered inference systems and also for paramodulation ([SL91] describe a lazy paramodulation calculus that is complete with set of support).

In some cases, depending on the syntactic properties of the given finite saturated set S, decision procedures for the theory are obtained. This is the case for instance for saturated sets of equations E, i.e., where saturation is in fact (unfailing) Knuth-Bendix completion, and where rewriting with E decides the word problem:

Theorem 7. *Let E be a consistent saturated (with respect to $\mathcal{H}$ and $\succ$) set of equations. Then, for all terms s and t, $E \models s \simeq t$ iff $t \Rightarrow_E \circ \Leftarrow_E s$.*

Decision procedures are also obtained for entailment problems $S \models G$, where S is saturated under $\mathcal{I}$ and G is a variable-free first-order formula, provided the ordering is such that for any ground term there are only finitely many smaller ground terms. In fact, to refute $S \cup \neg G$, only inferences in which one premise is ground (and initially from the set of clauses obtained from clausifying $\neg G$) can possibly be non-redundant. Any such inference produces a conclusion with an ordering constraint that bounds each term in the conclusion from above by a ground term in G. Therefore, clauses derived from non-redundant inferences have only finitely many ground instances where, in addition, all terms are smaller than or equal to a term in G. By a straightforward induction argument and by using König's Lemma, saturation will eventually terminate.

In particular if S is empty (hence trivially saturated), and if G is just a single ground Horn clause $s_1 \simeq t_1, \ldots, s_k \simeq t_k \rightarrow s \simeq t$ we obtain the so-called congruence closure problem of deciding whether or not $s_1 \simeq t_1, \ldots, s_k \simeq t_k \models s \simeq t$ is true in FOLE. For this special case one may choose an arbitrary reduction ordering for (finitely) saturating $\{s_1 \simeq t_1, \ldots, s_k \simeq t_k\} \cup \{s \simeq t \rightarrow\ \}$, provided one performs simplification by rewriting eagerly. In this special case, any superposition inference yields a conclusion that renders the maximal premise of the inference redundant. By introducing new constants for any of the subterms in G and by organizing rewrite rules into a union-find data structure, this saturation process can be organized in time $O(n \log n)$ [Kap97,BT00]

Other syntactic restrictions on non-equational saturated sets S that are quite easily shown to lead to decision procedures include *reductive* Horn clauses (also called conditional equations) or *universally reductive* general clauses [BG94a], even in cases where the reduction ordering is not isomorphic to ω. This applies to certain restricted kinds of queries only.

We will expand further on the subject of deciding entailment problems in the section 1.8 for the case of non-equational first-order theories.

1.6 Paramodulation with constrained clauses

In this section the power of using constrained clauses is further exploited. By keeping the equality constraints with the conclusion of an inference without solving them we can avoid further inferences on subterms of this conclusion which come from the unification problem. By keeping the ordering constraints we obtain a single constrained clause representing the exact set of ground instances of the conclusion that fulfill the ordering restrictions of the inference. Hence working with constrained clauses and inheriting these constraints in the inferences allows us to avoid many inferences and hence to further restrict the search space. To avoid certain technical complications, in this section we only consider Horn clauses. For the general case we refer to the overview articles [BG98] and [NR01].

1.6.1 Equality constraint inheritance: basic strategies

We now analyse the first constraint-based restriction of the search space: the so-called *basicness restriction*, where superposition inferences on subterms created by unifiers on ancestor clauses are not performed. This restriction can be conveniently expressed by inheriting the equality constraints without applying (or even computing) any unifiers. What is required is that satisfiability of certain equality constraints is checked. In fact, whenever a constrained clause with an empty skeleton is derived, only if the constraint is satisfiable have we found a contradiction.

In the following, we call a set of Horn clauses S with equality constraints *closed under* $\mathcal{H}$ *with equality constraint inheritance* if $D \mid T_1 \wedge \ldots \wedge T_n \wedge s \doteq t$ is in S whenever $C_1 \mid T_1 , \ \ldots , \ C_n \mid T_n$ are clauses in S and there is an inference by $\mathcal{H}$ with premises $C_1, \ldots, C_n$ and conclusion $D \mid s \doteq t \wedge OC$ and $T_1 \wedge \ldots \wedge T_n \wedge s \doteq t \wedge OC$ is satisfiable. In this setting, the ordering constraints OC associated with inferences are only checked for satisfiability but not inherited themselves.

This strategy is incomplete in general:

Example 4. Let $\succ$ be the lexicographic path ordering where $a \succ_{\mathcal{F}} b$. Consider the following unsatisfiable clause set S:

$$\begin{array}{ll} 1. & \rightarrow a \simeq b \\ 2. & \rightarrow P(x) \mid x \doteq a \\ 3. & P(b) \rightarrow \end{array}$$

S is clearly closed under $\mathcal{H}$ with equality constraint inheritance, since no inferences by $\mathcal{H}$ that are compatible with the constraint of the second clause can be made. We have $a \succ_{\mathcal{F}} b$ and hence the first clause could only be used by superposing a on some non-variable subterm, while superposition left (i.e., resolution) between 2 and 3 leads to a clause with an unsatisfiable constraint $x \doteq a \wedge b \doteq x$. However, S does not contain the empty clause. This incompleteness is due to the fact that the usual lifting arguments, like the ones in Theorem 2, do not work here, since they are based on the existence of *all* ground instances of the clauses. Note that this is not the case here: the only instance of the second clause is $P(a)$, whereas the lifting argument in Theorem 2 requires the existence of the instance $P(b)$. $\square$

Fortunately, the strategy is complete for *well-constrained* sets of clauses, which turn out to be adequate for many practical situations. We do not want to define this concept in detail. An important special case is when initial clauses do not carry any constraints, or when clauses are non-equational.

Theorem 8. *Let S be the closure under $\mathcal{H}$ with equality constraint inheritance of a well-constrained set S_0 of Horn clauses. Then either S_0 is satisfiable, or else S contains the empty clause (possibly as an instance of a constrained clause).*

We provide some hints about the proof for the case that clauses in S_0 carry no constraints so that contraints are only generated for clauses produced by inferences in $\mathcal{H}$. The main ideas about constructing candidate models and reducing counterexamples is the same as before. The only modification of the previous completeness proofs is with regard to the ground instances of S from which the rewrite system R is constructed. This time we construct R from the set GS of all ground instances $C\sigma$ of clauses $C \mid T$ in S which are *irreducible at substitution positions*. That means that if p is a position in C for which $C|_p$ is a variable x then $x\sigma$ is irreducible by all rules in $R_{C\sigma}$ (produced by irreducible instances of S smaller than $C\sigma$ in the clause ordering). (Technically this requires to define R and GS simultaneously by induction over the clause ordering.)

Now, clearly, if $C\sigma$ is reducible by a rule in $R_{C\sigma}$, the matching must take place at a non-variable position in the *skeleton* C of the constrained clause $C \mid T$. Ground inferences of $\mathcal{H}$ taking place at such positions in $C\sigma$ can easily be lifted to $C \mid T$. Therefore, if S is saturated under $\mathcal{H}$ with equality constraint inheritance, then GS is saturated under $\mathcal{H}$ without equality constraint inheritance. To see this, one also has to show that ground instances of clauses produced by an inference in $\mathcal{H}$ from premises in GS again belong to GS, that is, are themselves irreducible at substitution positions. This, however, is straightforward for Horn clauses.[6]

Now, by applying Theorem 2, we have shown that either GS is satisfiable or contains the empty clause. Moreover, if GS is satisfiable, then R^* is a model of GS. It remains to be shown that then R^* is also a model of S_0. S contains S_0 and S_0 has no constraints. If $C\sigma$ is ground instance of a clause C in S_0 consider the reduced instance $C\sigma'$ of C where $\sigma'(x)$ is the normal form of $\sigma(x)$ with respect to R. Obviously, $C\sigma'$ is in GS, and therefore true in R^*. As $C\sigma$ can be obtained back from $C\sigma'$ by replacing terms by congruent (with respect to R^*) terms, $C\sigma$ is also true in R^*.

This last step of showing that R^* is not only a model of GS, but also a model of S_0, fails, as shown in the example above, if clauses in S_0 may carry arbitrary constraints. The reduced substitution σ' might not be a solution of the constraint attached to C.

The proof for the case in which S_0 contains only non-equational clauses is even simpler. As there are no equations between non-atomic terms and no predicate variables that could be instantiated with atom terms, any ground instance of a constrained clause is irreducible.

1.6.2 Ordering constraint inheritance

The ideas on equality constraint inheritance apply as well to ordering constraint inheritance or to a combination of both:

A set of constrained Horn clauses S *closed under $\mathcal{H}$ with ordering constraint inheritance* if $D\sigma \mid T_1 \wedge \ldots \wedge T_n \wedge OC$ is in S whenever $C_1 \mid T_1$, $\ldots$, $C_n \mid T_n$ are clauses in S and there is an inference by $\mathcal{H}$ with (unconstrained) premises $C_1, \ldots, C_n$ and conclusion $D \mid s \doteq t \wedge OC$, $T_1 \wedge \ldots \wedge T_n \wedge s \doteq t \wedge OC$ is satisfiable, and $\sigma = mgu(s,t)$.

A set of constrained Horn clauses S *closed under $\mathcal{H}$ with equality and ordering constraint inheritance* if $D \mid T_1 \wedge \ldots \wedge T_n \wedge s \doteq t \wedge OC$ is in S whenever $C_1 \mid T_1$, $\ldots$, $C_n \mid T_n$ are clauses in S and there is an inference by $\mathcal{H}$ with premises $C_1, \ldots, C_n$ and conclusion $D \mid s \doteq t \wedge OC$ and $T_1 \wedge \ldots \wedge T_n \wedge s \doteq t \wedge OC$ is satisfiable.

The following theorems are proved as for the case of equality constraint inheritance:

Theorem 9. *Let S be the closure under $\mathcal{H}$ with ordering constraint inheritance of a well-constrained set S_0 of Horn clauses. Then either S_0 is satisfiable, or else S contains the empty clause (possibly as an instance of a constrained clause).*

---

[6] Here is where the case of general clauses becomes technically involved.

Theorem 10. *Let S be the closure under $\mathcal{H}$ with equality and ordering constraint inheritance of a well-constrained set S_0 of Horn clauses. Then either S_0 is satisfiable, or else S contains the empty clause (possibly as an instance of a constrained clause).*

1.6.3 Basic paramodulation

It is possible to further restrict the inference system $\mathcal{H}$ with constraint inheritance, at the expense of weakening the ordering restrictions. Roughly, the improvement comes from the possibility of moving the inserted right hand side in conclusions to the constraint part, thus *blocking* this term for further inferences. On the other hand, paramodulations take place only *with* the maximal term, like in superposition, but *on* both sides of the equation containing the maximal term. More precisely, the inference rule of (ordered, basic) paramodulation right then becomes:

ordered paramodulation right:

$$\frac{\Gamma' \rightarrow l \simeq r \qquad \Gamma \rightarrow s \simeq t}{\Gamma', \Gamma \rightarrow s[x]_p \simeq t \quad | \ x \doteq r \ \wedge \ s|_p \doteq l \ \wedge \ l > r, \Gamma' \ \wedge \ (s > \Gamma \vee t > \Gamma)}$$

where $s|_p$ is not a variable, and x is a new variable. The inference rule for paramodulation left is defined analogously. It is clear that these inference rules are an improvement upon superposition only when applied (at least) with inheritance of the part $x \doteq r$ of the equality constraint (since otherwise the advantage of blocking r is lost).

Basic paramodulation was introduced and proved complete in [BGLS95]. The completeness proof is an easy extension of the proof techniques that we have described above.

The completeness results for theorem proving strategies with constraint inheritance that we have formulated in this chapter do not speak about saturation and redundancy. In fact, we have deliberately ignored this subject here, as the technical details are quite involved. As we have indicated in the proof sketch for Theorem 8, constraint inheritance implies that deduction takes place only with reduced instances of clauses. Therefore any suitable notion of redundancy that is based on implication from smaller clauses will be admissible only if the smaller clauses are reduced themselves. The difficulty is that the rewrite system R with respect to which instances have to be reduced is not effectively known. In other words, irreducibility cannot be absolute but rather has to be relative. We again refer to the overview paper [NR01] for details on this subject.

1.7 Paramodulation with built-in equational theories

In principle, the aforementioned superposition methods apply to any set of clauses with equality, but in some cases special treatments for specific equational subsets of the axioms are preferable. On the one hand, some axioms generate many slightly different permuted versions of clauses, and for efficiency reasons it is many times better to treat all these clauses together as a single one representing the whole class. On the other hand, special treatment can avoid non-termination of completion procedures, as would occur with the equation $f(a,b) \simeq c$ in the presence of the laws of associativity and commutativity of f. Also, some equations like the commutativity axiom are more naturally viewed as "structural" axioms (defining a congruence relation on terms) rather than as "simplifiers" (defining a reduction relation). This allows one to extend completion procedures to deal with congruence classes of terms instead of single terms, i.e., working with a *built-in* equational theory E, and performing rewriting and completion with special E-matching and E-unification algorithms.

Special attention has always been given to the case where E consists of axioms of associativity (A) and commutativity (C), which occur very frequently in practical applications, and are well-suited for being built in due to their permutative nature.

Building permutativity into the data structure requires adapting the semantics of the symbol $>$ in the constraints. Now, the ordering $\succ$ has to be an *AC-compatible* reduction ordering (if $s' =_{AC} s \succ t =_{AC} t'$ then $s' \succ t'$) that is total on the AC-congruence classes of ground terms ($s \neq_{AC} t$ implies that either $s \succ t$ or $t \succ s$). Similarly $\doteq$ is interpreted as AC-equality. For simplicity, we will restrict ourselves again to Horn clauses, although all results can be extended to general clauses in the same way as before.

The full inference system for Horn clauses modulo AC, called $\mathcal{H}_{AC}$ includes the rules of $\mathcal{H}$ (under the new semantics for $\succ$ and $\doteq$) *plus* the following three specific rules:

AC-superposition right:

$$\frac{\Gamma' \to l \simeq r \qquad \Gamma \to s \simeq t}{\Gamma', \Gamma \to s[f(r,x)]_p \simeq t \quad \mid \; s|_p \doteq f(l,x) \;\wedge\; l > r, \Gamma' \;\wedge\; s > t, \Gamma}$$

AC-superposition left:

$$\frac{\Gamma' \to l \simeq r \qquad \Gamma, \underline{s \simeq t} \to \Delta}{\Gamma', \Gamma, s[f(r,x)]_p \simeq t \to \Delta \quad \mid \; s|_p \doteq f(l,x) \;\wedge\; l > r, \Gamma' \;\wedge\; s > t, \Gamma}$$

AC-top-superposition:

$$\frac{\Gamma' \to l' \simeq r' \qquad \Gamma \to l \simeq r}{\Gamma', \Gamma \to f(r',x') \simeq f(r,x) \quad \mid \; f(l',x') \doteq f(l,x) \;\wedge\; l' > r', \Gamma' \;\wedge\; l > r, \Gamma}$$

In these rules, where x and x' are new variables, the term l can be restricted to terms that carry the AC-symbol f at their top. This property is expressible in the constraint language and could be added to the constraint. The new variable x introduced in an inference by AC-superposition (right or left) is a so-called *extension variable* for f. AC-top-superposition is only needed if both l and l' have head symbol f. Finally, as in the non-AC case, the superposition inferences are needed only if $l|_p$ is non-variable, and AC-superposition (left or right) can be restricted to f positions p in l which do not occur as arguments of an f occurrence.

Theorem 11. *The inference system $\mathcal{H}_{AC}$ is refutationally complete for sets of Horn clauses with built-in AC-theories.*

1.7.1 Constraint inheritance and built-in theories

By inheriting equality constraints one can avoid some of the complexity of E-unification, which otherwise is one of the main drawbacks of working with built-in theories. In the AC-case there may be doubly exponentially many AC-unifiers for any two terms e.g. a minimal complete set for $x + x + x$ and $y_1 + y_2 + y_3 + y_4$ contains more than a million unifiers. Therefore the number of possible conclusions of an inference can be extremely high.

Theorem 12. *The inference system $\mathcal{H}_{AC}$ is refutationally complete with ordering constraint inheritance for well-constrained sets S of Horn clauses with built-in AC-theories.*

Again this result can be extended to general clauses using the same techniques as in the free case (see [NR97] for details).

As said before, by inheriting constraints, the computation of unifiers and the generation of multiple conclusions in an inference becomes unnecessary. But it is possible to go beyond. By using constraints it is possible to deal, in an effective way, with theories E with an *infinitary* E-unification problem, i.e. theories where for some unification problems any complete set of unifiers is infinite. This is the case of theories containing only associtivity axioms for some function symbols, which is developed in a similar way as done here for the AC-case in [Rub96].

Finally, by inheriting equality constraints one can consider any built-in theory E, even when the E-unification problem is undecidable, if an adequate inference system and ordering are found (although these ideas require a further development for concrete E). Incomplete methods can then be used for detecting cases of unsatisfiable constraints, and only when a constrained empty clause $\Box \mid T$ is found, one has to start (in parallel) a semidecision procedure proving the satisfiability of T. This method is interesting as well if decision procedure for the E-unification problem is available, since the incomplete methods can be more efficient and hence more efective in practice.

1.8 Effective saturation of first-order theories

Saturation up to redundancy also terminates for many consistent theories. This is particularly the case if ordering constraints are inherited throughout inferences and if sufficiently powerful techniques for simplification and the elimination of redundancies are employed. In this section we briefly describe two of the applications in which finitely saturated theories play a central rôle. We intend to demonstrate that saturation can be understood as a (partial) compilation process through which, when it terminates, certain objectives with regard to efficiency can be achieved in an automated way.

In this section we expand on the material provided in Section 1.5.1 above, but—for reasons the discussion of which is beyond the scope of these notes—restrict ourselves to non-equational clauses where superposition specializes to ordered resolution as defined in Section 1.4.7.

Remember that we formally view non-equational atoms $p(s_1, \dots, s_k)$ as special equations of the form $p(s_1, \dots, s_k) \simeq \top$, so that formally $p(s_1, \dots, s_k)$ and the s_j are terms, which we have called, respectively, atomic terms and non-atomic terms. In this section when we speak of an *atom*, we mean an atomic term, and when we speak of a *term* we mean a non-atomic term. Consequently, a *term ordering* will be any well-founded ordering on non-atomic terms, whereas an *atom ordering* will be any well-founded ordering on atomic terms. Also, atom orderings will usually be *compatible* with some term ordering in that $p(s_1, \dots, s_k) \succ q(t_1, \dots, t_n)$ in the atom ordering, whenever for all $1 \leq i \leq n$ there exists a $1 \leq j \leq k$ such that $s_j \succ t_i$ in the term ordering. A term s is said to *occur* in a clause C, if s occurs at an argument position of a predicate in C. Note that, by this definition, a proper subterm of a term occurring in C will not necessarily be considered as occurring in C.

1.8.1 Automated complexity analysis

Motivated by work on "local theories" by [McA93], in [BG96] the relation between saturation and decidability issues has been refined to complexity issues. It turns out that the complexity of the ordering used for saturation is directly related to the complexity of the entailment problem for a theory.

Let S be a set of clauses (with variables). The *(ground) entailment problem* for the *theory* S consists in checking whether or not a *query* C, where C is a variable-free clause, is logically implied by S. A local theory is one where this problem can be decided merely by considering ground instances of S derived, by means of a term ordering from the ground terms in S and C. In this section, a *term ordering* $\succ$ can be any partial, well-founded ordering on ground terms.

More precisely, suppose that the *complexity* of a term ordering $\succ$ can be bounded by functions f, g in that there are at most $O(f(n))$ ground terms smaller than or equal to some ground term in any given finite set of terms of size n,[7] and such that these smaller terms can be enumerated in time $O(g(n))$.

[7] This is the number of symbols needed to write down the terms in T in (non-sharing) prefix notation.

Given such an ordering $\succ$ of finite complexity and a set of ground terms T, by $T_{\preceq}$ we denote the set of ground terms smaller than (in $\succ$) or equal to a term in T. Moreover, given a set of ground terms T, by S_T we denote the set of ground instances of S that contain terms in T only. We call S (order-) *local* (with respect to $\succ$), if for each ground query C, $S \models C$ if, and only if, $S_{T_{\preceq}} \models C$, where T is the set of ground terms occurring in C or in S. Applying the result of Dowling and Gallier [DG84], if S contains m clauses and is local with respect to $\succ$—an ordering of complexity f, g—then the entailment problem for queries of size n is decidable in time $O(m * f(n)^k + g(n))$ where k depends only on the theory S. Often the theory S is considered a constant so that we obtain $O(f(n)^k + g(n))$, with k a constant, as an upper bound for the time complexity.

In particular, the entailment problem is of polynomial time complexity, if the ordering is of polynomial complexity. For the constant k one may take the *degree* of S, where the degree of a clause is the minimal number of different terms containing all the variables in the clause, and where the degree of a finite set of clauses is the maximum of the degrees of its element. Order locality extends the concept of locality as introduced by McAllester which appears as the special case in which the term ordering is the subterm ordering. It turns out there is a close relationship between order localiy and saturation with respect to ordered resolution.

Given a *total* term ordering $\succ$, we call S *saturated* (with respect to $\succ$) if S is saturated up to redundancy under ordered resolution with selection based on $\succ$ and with respect to some atom ordering $\succ_0$ that is compatible with $\succ$.[8] If the term ordering $\succ$ is only partial, we call N *saturated* (with respect to $\succ$) whenever N is saturated with respect to *every* total term ordering containing $\succ$.

Theorem 13 ([BG96]). *Suppose $\succ$ is a term ordering, and S is a set of Horn clauses that is saturated with respect to $\succ$. Then S is local with respect to $\succ$.*

Proof. (Sketch) Suppose C is a ground clause. Then we may define a total term ordering $\succ'$ extending $\succ$ such that $s \succ' t$ whenever s is a term for which there is no term t' in C such that $t' \succeq s$. Also, let $\succ_0$ be a total, well-founded atom ordering that is compatible with $\succ'$. By assumption, S is saturated under ordered (with respect to $\succ_0$) resolution with selection based on $\succ'$. Since S is saturated, inferences in which both premises are clauses in S are redundant. To decide whether the ground query C is entailed, one negates C,

[8] Selection of negative literals is said to be *based* on a term ordering $\succ$, whenever a negative literal is selected if, and only if, it contains the maximal term of the clause. One notices that with ordering-based selection, the particular choice of a compatible atom ordering $\succ_0$ does not affect the ordering constraints associated with the resolution inferences. The choice may have an effect on redundancy, though.

adds the resulting unit clauses to S, and saturates the resulting set. Clauses D generated by ordered inferences with one premise in S and one premise from $\neg C$ cannot generate clauses with a term not smaller in $\succ$ than, or equal to, some term in C. Atoms containing such terms would be larger in $\succ_0$ than any atom in $\neg C$. By induction this property is maintained for any clause derived subsequently from S, $\neg C$, and D. Thus, if C is a logical consequence of S, it already follows from the set of ground instances of S in which all terms are smaller than (in $\succ$), or equal to, a term in C.

Consequently, the entailment problem for a finite Horn theory S (considered constant) is of complexity $O(f^k + g)$, for some constant k, if S is saturated with respect to a term ordering of complexity f, g.

Example 5. The following represents a tractable (incomplete) fragment of propositional logic.

$$
\begin{aligned}
true(p) &\to true(or(p,q)) \\
true(p) &\to true(or(q,p)) \\
true(or(p,q)), true(not(p)) &\to true(q) \\
true(or(p,q)), true(not(q)) &\to true(p) \\
true(not(p)), true(not(q)) &\to true(not(or(p,q))) \\
true(not(or(p,q))) &\to true(not(p)) \\
true(not(or(p,q))) &\to true(not(q)) \\
true(not(p)), true(p) &\to true(\mathsf{ff}) \\
true(\mathsf{ff}) &\to true(p) \\
true(not(not(p))) &\to true(p) \\
true(p) &\to true(not(not(p)))
\end{aligned}
$$

This set of Horn clauses is saturated under all total and well-founded extension of the least partial ordering on terms satifying (i) $s \succ t$ if t is a proper subterm of s, (ii) $or(s,t) \succ not(s)$, (iii) $or(s,t) \succ not(t)$, and (iii) $s \succ \mathsf{ff}$, for $s \neq \mathsf{ff}$. This fact can be checked automatically by the Saturate system [GNN94]. In this partial ordering, any term s has at most linearly many (in the size of s) smaller terms. Hence ground entailment and, thus, provability for the encoded fragment of propositional logic, is decidable in polynomial time. In fact it is decidable in linear time since the degree of any clause is 1.

Sometimes the natural presentation of a theory is not saturated with respect to a desired ordering, but can be finitely saturated, see [BG96] for examples that have also been run on Saturate. In such cases saturation can be viewed as an optimizing compiler that adds sufficiently many "useful" consequences to a theory presentation so as to achieve a certain complexity bound for its entailment problem.

In order for saturation to terminate for these weak orderings (that is, to simultaneously saturate a theory with respect to all total, well-founded extensions of the partial ordering), ordering constraints have to be inherited throughout inferences (as explained in the section 1.6.2 for the general case of equational clauses). Moreover, redundancy elimination techniques have to be implemented. In this context it is important to note that while in the equational case the inheritance of ordering constraints comes at the expense of a weaker concept of redundancy, resolution with constraint inheritance is compatible with the standard notion of redundancy (as defined in the section 1.5).

1.8.2 Deduction with saturated presentations

Saturated presentations of a theory may give rise to improved inference systems for that theory. Again, deduction with constrained clauses is essential for both obtaining the saturated presentation as well as for defining the resulting specialized inference system.

The following constrained clause set T defines the transitive-reflexive closure p^* of a binary predicate p:

$$
\begin{array}{rll}
 & \rightarrow p^*(x,x) & \quad (1)\\
p(x,y) & \rightarrow p^*(x,y) & \quad (2)\\
p^*(x,y)\ ,\ p^*(y,z) & \rightarrow p^*(x,z) & \quad (3)\\
p^*(x,y)\ ,\ p(y,z) & \rightarrow p^*(x,z) & \mid\ y>z, y>x\ (4)\\
p(x,y)\ \ ,\ p^*(y,z) & \rightarrow p^*(x,z) & \mid\ x>y, x>z\ (5)\\
p^*(x,y)\ ,\ p(y,z) & \rightarrow p^*(x,z) & \mid\ z>x, z>y\ (6)\\
p(x,y)\ \ ,\ p(y,z) & \rightarrow p(x,z) & \mid\ y>x, y>z\ (7)
\end{array}
$$

The presentation is saturated under ordered resolution with selection. The ordering has to be chosen such that it is well-founded and total on ground terms and such that (i) $A \succ B$ whenever the maximal term in A is greater then the maximal term in B, and (ii) $p^*(s,t) \succ p(u,v)$, whenever the maximal term of s and t is the same as the maximal term of u and v. The selection function is assumed to select the maximal negative atom, if the maximal term of the clause occurs either in a negative p^*-atom or else in a negative p-atom but not in a positive p^*-atom. In all other cases, no atom is selected. Checking that the system is saturated is not difficult, but tedious. A sufficiently powerful saturation procedure such as the one implemented in the Saturate system [GNN94] will produce the set automatically from the first three clauses.[9]

[9] Clause (7) is not a consequence of (1)–(3). The actual clause obtained with Saturate was $p(x,y),\ p(y,z) \rightarrow p^*(x,z)\ \mid\ y>x, y>z$. But in the application we are interested in, non-theory clauses do not contain negative occurrences of p, so that the use of clause (7) is justified.

For instance, suppose

$$\frac{p(x,y),\ p^*(y,z) \to p^*(x,z) \ \mid \ x > y, x > z \quad p^*(x,z),\ p^*(z,u) \to p^*(x,u)}{p(x,y),\ p^*(y,z),\ p^*(z,u) \to p^*(x,u) \ \mid \ x > y, x > z, x \geq u}$$

is an inference by ordered resolution with selection (and with constraint inheritance) from the fifth and the third clause. Then $p^*(x,z)$ must be selected in the second premise, and therefore $x \succeq u$ must be true, as is reflected in a constraint in the conclusion. If $x = u$, then the conclusion of the inference is subsumed by clause (1). If $x \succ u$, and $x \succ y$ and $x \succ z$, then the conclusion follows from the two clauses $p^*(y,z),\ p^*(z,u) \to p^*(y,u)$ and $p(x,y),\ p^*(y,u) \to p^*(x,u)$, both of which are smaller than the main premise of the above inference. We may conclude that the inference is redundant.

Now consider the saturation of a clause set $T \cup N$ that contains T as a sub-theory. We may assume that atoms with predicate symbol p occur only positively, and atoms with p^* occur only negatively, in N. (Any negative [positive] occurrence of p [p^*] in N can be replaced by p^* [p] without affecting the satisfiability or unsatisfiability of $T \cup N$.) The following observations apply to ordered inferences in which one premise is from T and one from N.

(a) Clauses (3) or (4) can be ruled out as premises from T, because a (negative) atom with predicate p^* is selected in them, while N contains no positive occurrences of such atoms.

(b) Hyper-resolution inferences[10] with clauses in N as side premises (electrons) and clause (7) as main premise (nucleus) can be written as follows (in the ground case, with the theory clause omitted):

Ordered chaining, right

$$\frac{\Gamma \to p(s,t), \Delta \quad \Gamma' \to p(t,u), \Delta'}{\Gamma, \Gamma' \to p(s,u), \Delta, \Delta' \ \mid \ t > s, u \wedge p(s,t) > \Gamma, \Delta \wedge p(t,u) > \Gamma', \Delta'}$$

(c) In clauses (5) and (6) the positive atom is maximal. Ordered resolution with side premise (5) or (6) and a clause N as main premise, followed by resolving the resulting negative atom with predicate p by another clause in N, is captured by the following inferences:

Ordered chaining, left (I):

$$\frac{\Gamma \to p(s,t), \Delta \quad \Gamma', \underline{p^*(s,u)} \to \Delta'}{\Gamma, \Gamma', p^*(t,u) \to \Delta, \Delta' \ \mid \ s > t, u \wedge p(s,t) > \Gamma, \Delta}$$

Ordered chaining, left (II):

$$\frac{\Gamma \to p(t,s), \Delta \quad \Gamma', \underline{p^*(u,s)} \to \Delta'}{\Gamma, \Gamma', p^*(u,t) \to \Delta, \Delta' \ \mid \ s > t, u \wedge p(t,s) > \Gamma, \Delta}$$

[10] Hyper-resolution is resolution where simultanously *all* selected negative literals are resolved Hyper-resolution can be simulated by iterated ordered resolution with selection. The optimization in hyper-resolution over iterated ordered resolution is that the intermediate results need not be kept.

As before, selection is indicated by underlining, and nothing is assumed to be selected in clauses which do not have anything underlined.

Inferences involving clauses (1) or (2) need not be dealt with specifically.

In summary, when one extends ordered resolution with selection by ordered chaining one obtains a refutationally complete specialization of resolution for theories with transitive and reflexive relations p^* in which no resolution inferences with the transitivity axiom for p^* are required.[11] In [BG94b] specific methods from term rewriting have been employed to obtain closely related results. The above considerations demonstrate that saturation of a theory, which can be automated to a large extent, may produce a practically useful inference system, the completeness of which might otherwise require non-trivial meta-level arguments.

It seems that many other theories, including congruence relations, orderings, and distributive lattices, can be engineered in a similar way. This sheds some new light on how theory resolution can be efficiently implemented in practice for theories that can effectively be saturated in such a way that "problematic" clauses such as transitivity are eliminated to a large extent. Pre-saturation of theory modules contributes to an efficient handling of large, but structured theories. Similar ideas have been pursued in [Non95] in the context of theorem proving for modal logics, where specific saturations of the respective "background theories" form a core part of similar methods.

Acknowledgements. We are grateful to Chris Lynch for his detailed comments on a previous version of this paper.

References

[Bac88] Leo Bachmair. Proof by consistency in equational theories. In *Proceedings, Third Annual Symposium on Logic in Computer Science*, pages 228–233, Edinburgh, Scotland, 5–8 July 1988. IEEE Computer Society.

[BG94a] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[BG94b] L. Bachmair and H. Ganzinger. Rewrite techniques for transitive relations. In *Proc. 9th IEEE Symposium on Logic in Computer Science*, pages 384–393. IEEE Computer Society Press, 1994.

[BG96] D. Basin and H. Ganzinger. Complexity analysis based on ordered resolution. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pages 456–465. IEEE Computer Society Press, 1996.

[BT00] L. Bachmair and A. Tiwari. Abstract congruence closure and specializations. In *Proceedings of the Seventeenth International Conference on Automated Deduction (CADE-00)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 64–78, Berlin, 2000. Springer-Verlag.

[11] The chaining inferences encode certain ordered inferences with transitivity. The advantage over unordered hyper-resolution strategies with the transitivity clause is a better balancing between forward and backward computation.

[BG98] Leo Bachmair and Harald Ganzinger. Equational reasoning in saturation-based theorem proving. In W. Bibel and P. Schmitt, editors, *Automated Deduction: A Basis for Applications.* Kluwer, 1998.

[BGLS95] L. Bachmair, H. Ganzinger, Chr. Lynch, and W. Snyder. Basic paramodulation. *Information and Computation*, 121(2):172–192, 1995.

[Com90] Hubert Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.

[DG84] W. F. Dowling and J. H. Gallier Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 1:267-284, 1984.

[Fit90] Melvin Fitting. First-order logic and automated theorem proving. Springer-Verlag, 1990.

[GNN94] H. Ganzinger, R. Nieuwenhuis, and P. Nivela. The Saturate system. User manual maintained on the Web, 1994.

[Kap97] D. Kapur. Shostak's congruence closure as completion. In H. Comon, editor, *Proceedings of the 8th International Conference on Rewriting Techniques and Applications*, pages 23–37, 1997. Vol. 1232 of *Lecture Notes in Computer Science*, Springer, Berlin.

[McA93] D. A. McAllester. Automated recognition of tractability in inference relations. *J. Association for Computing Machinery*, 40(2):284–303, 1993.

[Nie93] Robert Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47:65–69, August 1993.

[Nie99] Robert Nieuwenhuis. Rewrite-based deduction and symbolic constraints. In Harald Ganzinger, editor, *Automated Deduction—CADE-16*, volume 1632 of *Lecture Notes in Artificial Intelligence*, pages 302–313, Berlin, 1997. Springer-Verlag.

[Non95] Andreas Nonnengart. *A Resolution-Based Calculus for Temporal Logics.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, December 1995.

[NR97] Robert Nieuwenhuis and Albert Rubio. Paramodulation with Built-in AC-Theories and Symbolic Constraints. *Journal of Symbolic Computation*, 23(1):1–21, May 1997.

[NR01] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning.* Elsevier Science Publishers (to appear), 2001.

[Rub96] Albert Rubio. Theorem proving modulo associativity. In *1995 Conference of the European Association for Computer Science Logic*, LNCS 1092, Paderborn, Germany, September 1996. Springer-Verlag.

[RW69] G. A. Robinson and L. T. Wos. Paramodulation and theorem-proving in first order theories with equality. *Machine Intelligence*, 4:135–150, 1969.

[SL91] Wayne Snyder and Christopher Lynch. Goal directed strategies for paramodulation. In Ronald V. Book, editor, *Rewriting Techniques and Applications, 4th International Conference*, LNCS 488, pages 150–161, Como, Italy, April 10–12, 1991. Springer-Verlag.