Saturation of first-order (constrained) clauses with the *Saturate* system

Pilar Nivela and Robert Nieuwenhuis*

1 Background

During the last years, several *saturation*-based theorem proving techniques for full first-order (*ordering and/or equality constrained*) clauses with equality have emerged.

Bachmair and Ganzinger ([BG91]) apply a *model construction* technique for proving the refutational completeness of *strict superposition* (paramodulation restricted to maximal terms of maximal equations of clauses). Their powerful *abstract redundancy notions* allow *redundant inferences* to be ignored and *redundant clauses* to be deleted without loosing completeness.

This makes it also possible to compute finite *saturated* sets of axioms. Using saturated sets, more efficient proof strategies become (refutationally) complete, like the set-of-support strategy, which is normally incomplete for ordered inference systems and also for equality clauses. Saturated sets fulfilling some syntactic properties even provide decision procedures, like rewrite proofs do in the equational case. Moreover, if a finite saturated set (not containing the empty clause) is obtained for a given theory, then its *consistency* has been proved (normally one can only prove *inconsistencies*).

Strict *basic* superposition, where no inferences have to be computed on subterms generated in previous inferences (like in basic *narrowing*), is proved complete in [NR92a] by means of *equality constraints* and independently also in [BGLS92]. In [NR92b] it is shown that by means of *ordering constrained clauses* one can restrict the search space even further without loosing completeness, by keeping the ordering restrictions of the inferences in constraints which are *inherited*.

As far as we know, no effort had been spent up to now on solving the problems that appear when putting this kind of techniques into practice: designing feasible redundancy provers that fit into the theoretical concepts (a key aspect of the system), choosing adequate inference systems for each case, designing fair control strategies and heuristics depending on whether the purpose is to find a saturated set or a refutation proof, how (and when) to do constraint solving in practice, etc.

*We would like to thank Harald Ganzinger for inviting us to the Max-Planck-Institut für Informatik, Saarbrücken, Germany, and for his encouragement to carry out this work during our half-year stay there. Permanent address of both authors: Technical University of Catalonia, Pau Gargallo 5, 08028 Barcelona, Spain. E-mail: {nivela,roberto}@lsi.upc.es.

These problems are addressed here in the context of their implementation in our *Saturate* system, written in Quintus-Prolog, a modular toolkit for experimenting with such methods. The system includes several inference calculi (including the ones for equality and ordering constrained clauses), as well as facilities for recovering proofs of axioms, for computing timing statistics, for choosing in a user-friendly way the different settings of the system (control strategies, redundancy provers, inference systems, amount of user interaction...), etc. All this is built by combining modules like constraint solvers or redundancy provers, starting from the abstract data types for elements like clauses, constraints or terms.

2 A short overview of the system

The system has been designed with the aim of being flexible both in its *use* and in its *extension/modification*. We first describe the main facilities for a flexible *use* of the system:

1. Settings of the system
The user can choose *any* subset of all parts of the system according to the problem at hand. These settings can be interactively modified at any moment, i.e. it is not necessary to fix them initially, although the user may define different own default settings.

Inference rules currently available are: *strict superposition left, strict superposition right, equality factoring, equality resolution, merging paramodulation, ordered factoring*. For instance, a minimal complete set of inference rules can be chosen according to the kind of axioms to be treated.

Constraint inheritance strategies can be added in order to restrict the search space: no constraint inheritance (then completeness is preserved under more powerful redundancy criteria), ordering constraint inheritance, equality constraint inheritance (basic strategies), or the combination of both. Clauses with unsatisfiable constraints are tautologies and can be eliminated. Also when no constraints are inherited, checking the ordering restrictions of each inference amounts to deciding the satisfiability of an ordering constraint (this is decidable for LPO, cf. [Com90] and also [NR92b] under extended signatures). The constraint solving module is therefore an important part of the system, which is also frequently used in redundancy proofs.

Marking strategies. In [BG91] their completeness is proved: one can mark an arbitrary negative equation in clauses and compute no inferences with the non-marked equations of such marked clauses (cf. also [NN91]).

Selection strategies: it is possible to choose between various (fair) selection strategies for determining at every stage of the saturation process which clause is used to compute inferences with.

Redundancy proofs: the amount of effort dedicated to redundancy proofs can be set depending on the purpose of the process (refutation or completion). If the purpose is to obtain a saturated system for a given theory, then probably more redundancy provers should be activated. Cf. the following section for more details on redundancy provers.

Output information: this allows to observe particular aspects of a saturation process without getting lost in a huge amount of information. For instance, if we are interested in the behaviour of a particular redundancy prover, a particular inference rule, or the number of consequences generated by a particular clause, then the active output messages can be set in such a way that only this information is provided.

Timing points: they exist for the important tasks of the system, and allow to determine e.g. the time spent on a particular inference rule, a redundancy prover, or on constraint solving, by just activating the corresponding subset of timing points. Timing statistics are displayed when a saturated system is obtained or when the user requires this explicitly.

2. Saturation of a set of axioms

A saturation process of the current data base of clauses can be initiated according to the previous options. It is possible to read in user axioms and also already saturated sets (generated and saved from previous runs of the system). Saturated sets will be treated by the system accordingly, i.e. no unnecessary inferences will be computed. The saturation process finishes when a saturated set of clauses is found (e.g. when the empty clause is generated). If the empty clause is not in the final set, then the *consistency* of the theory has been proved and the saturated set can be saved for further efficient theorem proving in this theory. Also the state of a saturation process can be saved between executions.

3. Proof recovering

The system displays a complete proof for a given clause number, starting from the initial axioms and indicating the intermediate steps.

4. Help facilities

There are also on-line help facilities about all the possibilities of the system.

The main facilities for a flexible *extension/modification* of the system are due to the fact that the system has been developed following the traditional principles of module abstraction.

Let us consider for instance the integration of a new inference rule into the system. First, the rule has to be expressed in terms of the operations of the already existing abstract data types, for clauses or equations, like `get_max_eq_in_succedent_clause( C, E )`, or `is_greater_eq( E1, E2 )`. After this, the rule, eventual messages, timing points, proof recovering information, etc. only have to be declared in one table. This updates the menus and all aspects of the control. Modifying the internal representation of a data type needs of course only local changes.

3 Redundancy provers and examples

One of the limits of the current theorem proving systems is caused by the retention of too many clauses. Many theorem proving systems apply heuristics to decide which clauses are generated first, which ones are kept, and which ones are disregarded. However, in our opinion it is also worthwhile to investigate the practical applicability of the existing restrictive ordered inference systems and the powerful *abstract redundancy notions* for inferences and clauses in which completeness is not sacrificed *a priori.*

A clause is redundant (in a set of clauses S) if all its ground instances can be (semantically) deduced from *smaller* ground instances of clauses in S. Similarly, an *inference* is redundant if, for all its ground instances, its conclusion follows from instances smaller than the maximal premise. It is of course not possible to directly implement these abstract redundancy notions. Instead one should look for practical, sufficiently general methods that fit into these abstract notions. We have been able to do so and obtain some full first-order saturated systems that had not been obtained before by means of practical methods.

Classical redundancy proofs. Among the more classical methods, the system of course includes elimination of redundant literals, tautology detection, subsumption, condensement and demodulation. For clauses with equality, tautology checking amounts to deciding whether the succedent is deducible from the equations in the antecedent. We do this by rewriting the skolemized succedent with the (ground) completed skolemized antecedent. This method is also applied in the elimination of redundant literals. If for a clause C there exists a substitution σ such that $C\sigma \subseteq C$ then it is said that C is *condensed* into $C\sigma$. For example $p(x,y), p(a,b) \rightarrow q(a), q(x)$ is logically equivalent to (and can be replaced by) the smaller clause $p(a,b) \rightarrow q(a)$.

Case analysis on variables is applied in [MN90] for proving the ground confluence of ordered rewrite systems. For example, adding to the axioms for associativity and commutativity of $+$ the consequence $x + (y + z) = y + (x + z)$ yields a ground confluent ordered rewrite system. This can be shown by analyzing all possible orderings (with $=$ and $\succ$) between $x\sigma$, $y\sigma$ and $z\sigma$: $x\sigma = y\sigma = z\sigma$, $x\sigma = y\sigma \succ z\sigma$, $x\sigma \succ y\sigma = z\sigma$, etc. This method is extended in the *Saturate* system for proving redundancy properties on all ground instances of general clauses. However, this method has two main drawbacks: on one hand, it is expensive. The number of $f(n)$ of possible orderings for n variables is given by $f(3) = 13$, $f(4) = 75$, $f(5) = 541$, $f(6) = 4683, \ldots$ On the other hand, this case analysis is not always powerful enough: Suppose we have the equation $(x + y) + z = z + (x + y)$. Then the critical pair $(z + (x + y)) + u = u + ((x + y) + z)$ can be eliminated only by considering a case analysis on $x\sigma + y\sigma \succ z\sigma$, $x\sigma + y\sigma = z\sigma$, and $z\sigma \succ x\sigma + y\sigma$.

Constrained rewriting on ordering constrained equations $t = t' \, [\![C]\!]$, where $t = t'$ is an equation and C is an LPO-ordering constraint. If all instances of $t = t'$ for which C is satisfied are of the form $s = s$ then $t = t' \, [\![C]\!]$ is a tautology (this is decidable).

Let $u = v$ and $e \, [\![C]\!]$ be an equation and a constrained equation whit $e|_p = u\sigma$. Then $e \, [\![C]\!]$ rewrites by our notion of *constrained rewriting* into $e[v\sigma]_p \, [\![C \wedge u\sigma \succ v\sigma]\!]$ and into the *complementary* equation $e \, [\![C \wedge u\sigma \not\succ v\sigma]\!]$. If we apply such constrained rewrite steps whenever $C \wedge u\sigma \succ v\sigma$ is satisfiable in the sense of [NR92b], then this rewriting process terminates and provides a redundancy method that can handle examples like the previous one, as the search is more directed towards the different cases that have to be considered.

Clausal rewriting applying clauses of a set S, denoted $\rightarrow_S$, is defined on sets of clauses as follows: if C is a tautology or C is subsumed by a clause in S then $N \cup \{C\} \rightarrow_S N$. If $t = t' \vee l_1 \vee \ldots \vee l_n$ is a clause in S, and $C|_p = t\sigma$, then $N \cup \{C\} \rightarrow_S N \cup \{\, C[t'\sigma]_p \vee l_1\sigma \vee \ldots \vee l_n\sigma,\;\; C \vee \neg l_1\sigma,\; \ldots,\; C \vee \neg l_n\sigma \,\}$.

In a similar way, not only equality literals $t = t'$, but also positive and even negative atoms can be applied in clausal rewrite steps. In this generalization of demodulation, if $\{C\} \rightarrow_S^* \emptyset$ then $S \models C$. It is checked that the instances of the clauses of S used are smaller than C.

By this redundancy method and some restrictions of it, the *Saturate* system automatically obtains, among many other examples, a saturated set for total orderings:

$$\begin{array}{rcl@{\qquad\qquad}rcl} & \rightarrow & p(x,x) & & \rightarrow & p(x,y) \vee p(y,x) \\ p(x,y) \wedge p(y,z) & \rightarrow & p(x,z) & p(x,y) \wedge p(y,x) & \rightarrow & x = y \end{array}$$

References

[BG91] Leo Bachmair and Harald Ganzinger. Rewrite-based equational theorem proving with selection and simplification. Technical Report MPI-I-91-208, Max-Planck-Institut für Informatik, Saarbrücken, August 1991. To appear in Journal of Logic and Computation.

[BGLS92] Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation and superposition. In Deepak Kapur, editor, *11th International Conference on Automated Deduction*, LNAI 607, pages 462–476, Saratoga Springs, New York, USA, June 15–18, 1992. Springer-Verlag. To appear in Information and Computation.

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

[MN90] Ursula Martin and Tobias Nipkow. Ordered rewriting and confluence. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction*, LNAI 449, pages 366–380, Kaiserslautern, FRG, July 24–27, 1990. Springer-Verlag.

[NN91] Robert Nieuwenhuis and Pilar Nivela. Efficient deduction in equality horn logic by horn-completion. *Information Processing Letters*, 39(1):1–6, July 1991.

[NR92a] Robert Nieuwenhuis and Albert Rubio. Basic superposition is complete. In B. Krieg-Brückner, editor, *European Symposium on Programming*, LNCS 582, pages 371–390, Rennes, France, February 26–28, 1992. Springer-Verlag.

[NR92b] Robert Nieuwenhuis and Albert Rubio. Theorem proving with ordering constrained clauses. In Deepak Kapur, editor, *11th CADE*, LNAI 607, pages 477–491, Saratoga Springs, New York, 1992. Extended version to appear in Journal of Symbolic Computation.