System description: an equational constraints solver

Nicolas Peltier

Laboratory LEIBNIZ-IMAG
46, Avenue Félix Viallet 38031 Grenoble Cedex FRANCE
Nicolas.Peltier@imag.fr
Phone: (33) (0)4 76 57 48 05

1 Equational formulae: definition and solvability

Equational formulae are first-order formulae containing only "=" as a predicate symbol. A substitution σ is a *solution* of an equational formula $\mathcal{F}$ iff $\mathcal{F}\sigma$ is valid in the finite tree algebra (i.e. when = is interpreted as the syntactic equality on the Herbrand universe). Equational formulae have many applications in the domains of Automated Deduction, Artificial Intelligence and Computer Science (program verification [6], negation in logic programming [5, 14, 6, 3], inductive proofs [7], model building [4, 1, 20] etc.). They have been studied by many authors for several years. In particular, the validity problem for equational formulae in the empty theory has been proven to be decidable [17, 13, 16, 11].

Example 1. Let $\Sigma = \{0, succ\}$. $\forall z.x \neq succ(z) \wedge succ(y) = succ(succ(x))$ is an equational formula. This formula has a unique solution : $\{x \rightarrow 0, y \rightarrow succ(0)\}$.

In this paper, we describe a system, called $\text{Ecs}_{\text{ATINF}}$, able to solve equational formulae. $\text{Ecs}_{\text{ATINF}}$is a part of the Inference Laboratory ATINF developed in the LEIBNIZ laboratory since 1985 (see [2] for more details about ATINF). It is also a key component of our system for simultaneous search for refutations and models $RAMC_{\text{ATINF}}$[1]. To the best of our knowledge $\text{Ecs}_{\text{ATINF}}$ is the only system for solving equational constraints in the public domain.

2 Solving equational formulae

Given a sorted signature Σ and an equational formula $\mathcal{F}$, $\text{Ecs}_{\text{ATINF}}$ automatically computes the set of solutions of $\mathcal{F}$. For doing that, it uses the quantifier-elimination algorithm presented in [11, 6] in order to transform first-order formulae into a disjunction of formulae in a so-called definition with constraints solved form, from which the solution of the initial formula can be easily extracted. The notion of definition with constraints can be seen as the analogous of the notion of most general unifiers for unification problems: it gives the general form of the solutions of the initial formula.

Definition 1. A formula $\mathcal{F}$ is in *definition with constraints* solved form iff it is $\bot$ of if it is of the form

$$\exists u_1, \ldots, u_k.(\bigwedge_{i=1}^{n} x_i = t_i \wedge \bigwedge_{i=1}^{m} y_i \neq s_i)$$

where the x_i's and y_i's are variables, for all $i \leq n$, x_i occurs only once in $\mathcal{F}$, for all $i \leq m$, y_i does not occurs in s_i, y_i is not sort-restricted.

In [11], it is shown that any formula in definition with constraints solved form distinct from $\perp$ as a solution. Moreover, an algorithm is given for enumerating the solutions of a formula in definition with constraints solved form.

The algorithm transforming each first-order formula into a definition of constraints is given in [11]. It is specified as a set of rewriting rules, containing in particular the usual unification rules (decomposition, occur check, clash, replacement), specific rules for eliminating quantifiers (universality of parameter and explosion rules) and some boolean transformation rules (for example distributivity rules). We do not recall here the definition of these rules (the interested reader can consult [11]). Ecs$_{\text{ATINF}}$ uses some special strategies increasing the efficiency of the system. In particular, the application of the costly explosion and distributivity rules are *delayed*, and if possible *avoided*.

Roughly speaking, the resolution process is divided into two steps. During the first one, Ecs$_{\text{ATINF}}$ tries to eliminate the universal quantifiers occurring in the formula. For doing that it uses a standard unification algorithm for solving unification problems occurring in the formula, then it applies the universality of parameters rules. Finally, the distributivity and explosion rules are applied. The distributivity rule allows one to put the formula in conjunctive normal form (which is necessary for the elimination of universal quantifiers) and the explosion rule is needed for eliminating parameters occurring at non root positions in disequations. During the second step, the system uses the unification and explosion of disjunction rules to transform the formula into a disjunction of definition with constraints. Boolean transformation rules are used to simplify the formula at both steps.

3 Using Ecs$_{\text{Atinf}}$

The syntax used by our system is very close to the one of the theorem prover Otter [18] (which is very well known and widely used in Automated Deduction). Some further features (such as the use of sort symbols) are allowed for the definition of the formula.

Constant or function symbols are defined using the command: `declare(s(f(s1,s2,...,sn)))`, where $s_1, \ldots, s_n, s$ are sort symbols and f is a symbol of profile $s_1, \ldots, s_n \rightarrow s$. Formulae are defined using the command: `formula(name(x1,...,xn))` (where `name` is the name of the formula, and `x1,...,xn` the free variables) followed by the formula itself. Previously defined formula can be reused in defining a new formula : for example the commands `formula(p(x)). (x = a).` and `formula(q). (all y. p(y))` define a formula $p = \forall y.(y = a)$[1]. The symbols $\vee, \wedge, \Rightarrow, \Leftrightarrow$ are respectively denoted by `|, &, ->, <->`. Universal and existential quantified formulae are denoted by `(all x1,...,xn F)` and `(exists x1,...,xn F)`. $\neq$ is denoted by `!=`. The command `solve(name)` transforms the formula `name`

[1] This feature is particularly useful for model verification applications: it allows in particular to use Ecs$_{\text{ATINF}}$ to evaluate formulae in eq-interpretation, i.e. in Herbrand interpretations defined by equational formulae. See [4, 20] for details.

into a disjunction of definition with constraints. Some flags and parameters allow the user to control some parts of the resolution process.

ECS$_{\text{ATINF}}$ has been coded in C. It can be freely downloaded by the WEB, from http://leibniz.imag.fr/ATINF/Nicolas.Peltier/. It is self documented and examples of input files and experimental results are included.

Example We give below an example of problem that can be treated by our system.

```
% functional symbol declaration (optional)
declare(top(a)).
declare(top(f(top))).
declare(top(g(top,top))).

signature = { a,f,g }. % definition of the signature

formula(prob).   % formula to solve
(f(A) =f(B) | (all Z. ((Z = a) | (A != g(Z,Z))))).

solve(prob).

% Definition with constraint solved form

prob = (exists T,U,S ((-(T = U) & (A = g(U,T)))
| (A = g(U,a)) | (A = a) | (A = f(S)) | (A = B)))

Run Time: 0.01
```

4 Future work

Numerous extensions have been proposed to the algorithm in [11]. Algorithms for solving equational formulae in some particular non empty theories have been proposed and some extensions of the constraint language have been considered. In particular, algorithms have been proposed for shallow theories [10], ground equations [8], terms with integer exponents [19], membership constraints [9], some classes of higher order formulae [15], ordering constraints [12] etc. Some of these extensions could be included in our software in the future. In particular we plan to implement the algorithm for solving constraints on terms with integer exponents presented in [19].

We have proposed in [20] some improvements to the constraints solving algorithm of [11]. These improvements consist mainly in a new more suitable definition of "solved form" (allowing in particular the occurrence of some universal quantifiers in the definition with constraints) and a more careful control on the application of the key rules of distributivity and explosion. These extensions will also be included into our system in the near future.

References

1. C. Bourely, R. Caferra, and N. Peltier. A method for building models automatically. Experiments with an extension of Otter. In *Proceedings of CADE-12*, pages 72–86. Springer, 1994. LNAI 814.
2. R. Caferra and M. Herment. A generic graphic framework for combining inference tools and editing proofs and formulae. *Journal of Symbolic Computation*, 19(2):217–243, 1995.
3. R. Caferra and N. Peltier. Decision procedures using model building techniques. In *Proceeding of Computer Science Logic, CSL'95*, pages 130–144. Springer, LNCS 1092, 1996.
4. R. Caferra and N. Zabel. A method for simultaneous search for refutations and models by equational constraint solving. *Journal of Symbolic Computation*, 13:613–641, 1992.
5. A. Colmerauer. Equations and inequations on finite or infinite trees. In *FGCS'84*, pages 85–99, November 1984.
6. H. Comon. *Unification et Disunification. Théorie et Applications.* PhD thesis, INPG, Grenoble, 1988.
7. H. Comon. Inductive proofs by specification transformation. In *Proc. of RTA 89*. Springer, 1989.
8. H. Comon. Disunification: a survey. In Jean-Louis Lassez, Gordon Plotkin, editors. Computational logic: Essays in Honor of Alan Robinson. MIT Press, 1991.
9. H. Comon and C. Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, August 1994.
10. H. Comon, M. Haberstrau, and J.-P. Jouannaud. Syntacticness, cycle-syntacticness and shallow theories. *Information and Computation*, 111(1):154–191, May 1994.
11. H. Comon and P. Lescanne. Equational problems and disunification. *Journal of Symbolic Computation*, 7:371–475, 1989.
12. H. Comon, R. Nieuwenhuis, and A. Rubio. Orderings, AC-theories and symbolic constraint solving. pages 375–385, 1995.
13. K. Kunen. Answer sets and negation as failure. In J. L. Lassez, editor, *Proceeding of the Fourth International Conference on Logic Programming*, pages 219–227. The MIT Press, May 1987.
14. D. Lugiez. A deduction procedure for first order programs. In F. Levi and M. Martelli, editors, *Proceedings of the sixth International Conference on Logic Programming*, pages 585–599. The MIT Press, July 1989.
15. D. Lugiez. Positive and negative results for higher-order disunification. *Journal of Symbolic Computation*, 1995.
16. M. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 248—357. IEEE Computer Society, 1988.
17. A. Mal'cev. *The Metamathematics of Algebraic Systems: Collected Papers 1936–1967*, chapter Axiomatizable classes of locally free algebra of various type, pages 262–281. Benjamin Franklin Wells editor, North Holland, 1971. chapter 23.
18. W. W. McCune. *OTTER 3.0 Reference Manual and Guide*. Argonne National Laboratory, 1994.
19. N. Peltier. Increasing the capabilities of model building by constraint solving with terms with integer exponents. *Journal of Symbolic Computation*, 24:59–101, 1997.
20. N. Peltier. *Nouvelles Techniques pour la Construction de Modèles finis ou infinis en Déduction Automatique*. PhD thesis, Institut National Polytechnique de Grenoble, 1997.