System Abstract: E 0.3

Stephan Schulz

Institut für Informatik, Technische Universität München,
D-80290 München, Germany, `schulz@informatik.tu-muenchen.de`

Abstract. We describe the main characteristics of version 0.3 of the E equational theorem prover. E is based on superposition and rewriting. It features a powerful interface for specifying search guiding heuristics. We discuss some important details of the implementation and demonstrate the performance of the prover by presenting experimental results on the TPTP. Finally, we describe our future plans for improving the system.

1 Introduction

E is a fully automatic theorem prover for clausal logic with equality. It is sound and, if used with fair strategies, complete.

We had two aims in designing and implementing this new prover. First, we were looking for a new platform to continue our work on proof analysis and learning. While DISCOUNT [DKS97], the system used for our previous work, has been surprisingly adaptable, the code base was showing its age, and each new extension became more and more difficult. Moreover, DISCOUNT is restricted to unit equational proof problems, while most current application problems are more naturally encoded in full clausal logic.

Secondly, we wanted to improve the relatively poor performance of SETHEO [MIL+97] on problems with equality. The METOP calculus [Mos96] was designed to allow efficient handling of equality in a model-elimination prover. It requires a saturation phase to be interleaved with the top-down proof search.

Thus, E was designed as an efficient and flexible inference engine that enables us to easily implement the necessary features. In this paper we concentrate on the use of E as a stand-alone prover and only briefly mention special features resulting from the intended use of E within a METOP-based prover.

2 Calculus

E is a purely equational theorem prover, based on ordered paramodulation and rewriting. As such, it implements an instance of the superposition calculus described in [BG94]. We give a very short introduction to the calculus, discussing special aspects of E.

$Term(F, V)$ denotes the set of (first order) *terms* over a set F of function symbols and set V of variables. We write $t|_p$ to denote the subterm of t at a position p and write $t[p \leftarrow t']$ to denote t with $t|_p$ replaced by t'. An equation

$s = t$ is an (implicitly symmetrical) pair of terms. A positive literal is an equation $s = t$, a negative literal is a negated equation $s \neq t$. Literals can be represented as multi-sets of multi-sets of terms, with $s = t$ represented as $\{\{s\}, \{t\}\}$ and $s \neq t$ represented as $\{\{s, t\}\}$. A *ground reduction ordering* $>$ is a Noetherian partial ordering that is stable w.r.t. the term structure and substitutions and total on ground terms. $>$ can be extended to an ordering $>_1$ on literals by comparing the multi-set representation of literals with $\ggg\ggg$ (the multi-set-multi-set extension of $>$). Alternatively, we define $>_2$ on literals as $>_1 \setminus (\{(s = t, s' \neq t') | s, t, s', t' \in Term(F, V)\} \cup \{(s \neq t, s' = t') | s, t, s', t' \in Term(F, V)\})$, i.e. by making positive and negative literals incomparable.

By default, E implements the generating inference rules *Equality Resolution*, *Superposition (left/right)* and *Equality Factoring* as described in [BG94], i.e. restricted to maximal terms (w.r.t. $>$) and maximal literals (w.r.t. $>_1$).

From the many possibilities of simplifying or eliminating clauses in the superposition calculus, E implements *unconditional rewriting*, *subsumption*, *deletion of tautologies*, and a new contracting inference rule named *Simplify-Reflect* that allows us to directly eliminate negative literals that are subsumed by positive unit clauses, but cannot be rewritten:

$$\text{(SR)} \quad \frac{\mathtt{s = t} \qquad \mathtt{u[p \leftarrow \sigma(s)] \neq u[p \leftarrow \sigma(t)] \vee R}}{\mathtt{s = t} \qquad \mathtt{R}} \qquad \begin{array}{l}\text{if } \sigma(s), \sigma(t) \text{ are} \\ >\text{-incomparable}\end{array}$$

As (SR) only removes clauses that are *composite* with respect to the remaining set of clauses, the calculus is complete. For the case of unit clauses, it degenerates into *unfailing completion* [BDP89] as implemented in DISCOUNT.

In addition to the above general saturation strategy, E also implements the positive unit strategy for Horn clauses described in [Der91]. This is achieved by substituting $>_1$ with $>_2$ in the generating inference rules and by restricting the superposition rule to paramodulate only from positive unit clauses and into negative literals and positive unit clauses. As a third option, an intermediate strategy (paramodulating only from positive unit clauses, but into arbitrary literals) is available, and has been quite successful in the experimental evaluation.

Contrary to e.g. SPASS [WGR96], E does not implement special rules for non-equational literals or sort theories, as we expect this part to be taken care of by SETHEO in a later combined system. Instead, non-equation literals are encoded as equations and dealt with accordingly.

3 Search Control

The basic proof procedure of E is straightforward: The set of all clauses is split into a set of *processed* clauses and a set of *unprocessed* clauses. Initially, all input clauses are unprocessed, and the set of processed clauses is empty. The prover selects an unprocessed clause, simplifies it w.r.t. to the processed clauses, then uses it to simplify the processed clauses in turn. It then performs equality factoring, equality resolution and superposition between the selected clause and the set of processed clauses. The generated clauses are added to the set of unprocessed clauses. Processed clauses in which maximal terms have been subject to simplification inferences are also moved to the set of unprocessed clauses for reconsideration. The process stops when the empty clause is derived or no further inferences are possible.

There are two main parameters to the proof procedure: The reduction ordering and the order in which unprocessed clauses are selected. E currently implements KBO and LPO, with a variety of simple schemes to generate precedences and weights. A simple default is used unless otherwise specified.

The order in which unprocessed clauses are selected is determined by a search control heuristic. Such a heuristic sets up a variety of priority queues and a weighted round-robin scheme that determines from which queue the next clause is to be selected. Order within each queue is determined, in this order, by a *priority function* that can e.g. prefer all-negative clauses, ground clauses, or initial axioms, and by an *evaluation function* that is typically based on symbol counting. Completeness of the prover can be guaranteed by careful selection of priority and evaluation functions, or by simple addition of a fair queue (e.g. a FIFO-queue). Search control heuristics can be chosen from a predefined set or completely specified at run time. The prover also supports a simple automatic mode in which a heuristic is chosen based on some simple properties of the problem.

4 Implementation

E is build on top a layer of libraries that offer successively more specialized services, from low level I/O to the main proof search algorithm. The most outstanding feature probably is the use of perfectly shared terms. Except for short-lived temporary copies, each distinct subterm is represented exactly once in E. This means that each individual rewriting step potentially affects a large number of terms, literals and clauses. As an example, a single application of the equation $f(x) = x$ to any subterm of the form $f(a)$ simultaneously transforms $g(f(a), f(f(f(a)))$ into $g(a, a)$. This is achieved by recursively propagating the replacing of a term to all its superterms while ensuring that multiple identical copies created by this replacing collapse onto a single instance. Term positions on which rewriting is restricted by the calculus are kept distinct from normal terms and are thus exempt from this behaviour. Sharing of identical subterms reduces the number of allocated term cells by at least one order of magnitude, and allows very efficient detection of terms already in normal form with respect to a given set of rewrite rules.

The second important feature of the implementation is the use of *perfect discrimination trees* [Gra95] for rewriting, unit-subsumption and *Simplify-Reflect*. The performance of the discrimination trees is further improved by the use of age and size constraints on the branches of the tree. This allows us to backtrack from branches that cannot lead to a successful match very early.

E is implemented in ANSI-C. It has been successfully compiled and installed under various versions of GNU/Linux, SunOS, Solaris and HP-UX, using both the GNU C compiler and SUN's proprietary C compiler.

5 Experimental Results

As a result of the powerful way in which search control heuristics can be specified in E, the space of possible parameters is even bigger than for many other fully automatic theorem provers. We will therefore only present results of E in *fully automatic* mode, i.e. in the way a naive user would use the system. For comparison, we include data for SPASS 0.85 and Otter [MW97] on comparable hardware and with similar time limits[1].

We used the set of all clause normal form problems from the TPTP [SSY94] problem library, version 2.1.0, for evaluation. Results are presented for the complete set and for subsets described by the structure of the clauses and the equality content of the problem.

	No Equality			Equality			Overall
	Unit	Horn	Non-Horn	Unit	Horn	Non-Horn	
Problems	11	557	766	402	321	1218	3275
	Proofs						
E 0.31a	8	461	286	312	252	301	1620
Otter/Mace	8	448	294	289	207	276	1522
SPASS 0.85	8	434	349	261	201	350	1603
	Models						
E 0.31a	3	5	48	1	3	3	63
Otter/Mace	3	21	97	10	8	3	142
SPASS 0.85	3	5	110	1	0	3	122
	Total successes						
E 0.31a	11	466	334	313	255	304	1683
Otter/Mace	11	469	391	299	215	279	1664
SPASS 0.85	11	439	459	262	201	353	1725
	Average time (seconds/solution)						
E 0.31a	0.03	12.63	18.60	15.06	29.34	22.45	18.49
Otter/Mace	0.73	8.26	6.52	3.01	6.71	13.33	7.51
SPASS 0.85	0.01	8.34	20.29	14.26	26.57	15.25	15.90

Results for E have been obtained with the latest version, 0.31a, and in compliance with the guidelines for use of the TPTP. TPTP input files were unchanged except for removal of equality axioms and syntax transformation. The performance was evaluated on a cluster of SUN Ultra 60 workstations running at 300 MHz. CPU time per attempt was limited to 300 seconds, memory to 192 MB. Results for SPASS have been obtained on Pentium-II computers running at 300 MHz, and with a time limit of 300 seconds. Otter results are for the combination of Otter 3.0.5 and Mace 1.3.2 that participated in the CASC-15 ATP competition, running on Pentium-II computers at 400 MHz, and with a cut-off time of 300 seconds (as opposed to the 600 seconds on the Otter web page). In our experience, a 400 MHz Pentium-II computer performs within 5% of our SUN workstations for measuring CPU times on theorem proving tasks.

[1] We thank Bill McCune, Geoff Sutcliffe and Christoph Weidenbach for their assistance with obtaining and interpreting the data.

As the table shows, E finds slightly more proofs than either of the other two provers over the complete TPTP. It performs particularly well for unit problems, which are mostly pure equational, and Horn problems, where the three different saturation strategies complement each other well. E is slightly less successful for general clauses, especially in the non-equational case. We assume that the main reason for this is the lack of specialized inference rules for non-equational literals, which play an important role most general problems even if equality is present. Finally, E is not very powerful for showing satisfiability. This is not surprising, as we have made no attempt to improve the calculus in this direction.

6 Future Work

We believe that the results in the previous section show the potential of our implementation. However, a lot of work remains to be done. In particular, we want to create tools for proof presentation and analysis. We also will improve on the selection of evaluation functions, including goal-oriented and learning heuristics analogous to those implemented in DISCOUNT, and will automate the process of generating and/or selecting suitable term orderings.

Finally, we plan to integrate E and SETHEO to form a METOP-based hybrid system combining the strengths of the top-down and the bottom-up approach.

References

[BDP89] L. Bachmair, N. Dershowitz, and D.A. Plaisted. Completion Without Failure. *Coll. on the Resolution of Equations in Algebraic Structures, Austin, 1987*. Academic Press, 1989.

[BG94] L. Bachmair and H. Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 3(4):217–247, 1994.

[Der91] N. Dershowitz. Ordering-Based Strategies for Horn Clauses. *12th IJCAI, Sydney*, pp. 118–124. Morgan-Kaufmann, 1991

[DKS97] J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT: A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

[Gra95] P. Graf. *Term Indexing*. LNAI 1053. Springer, 1995.

[MIL+97] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. SETHEO and E-SETHEO – The CADE-13 Systems. *Journal of Automated Reasoning*, 18(2):237–246, 1997.

[Mos96] M. Moser. *Goal-Directed Reasoning in Clausal Logic with Equality*. CS Press, München, 1996. Ph.D. Thesis, Fakultät für Informatik, TU München.

[MW97] W.W. McCune and L. Wos. Otter: The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.

[SSY94] G. Sutcliffe, C.B. Suttner, and T. Yemenis. The TPTP Problem Library. *Proc. CADE-12, Nancy*, LNAI 814, pp. 252–266. Springer, 1994.

[WGR96] C. Weidenbach, B. Gaede, and G. Rock. SPASS & FLOTTER Version 0.42. *Proc. CADE-13, New Brunswick*, LNAI 1104, pp. 141–145. Springer, 1996.

The E 0.3 distribution is available on the WWW at
`http://wwwjessen.informatik.tu-muenchen.de/~schulz/WORK/eprover.html`