ANR-08-DEFI-005
DECERT

Task 3: Expressiveness
Report on strategies for efficient deduction

The Decert Consortium

January 5, 2010

1 Introduction

This task is devoted to making progress in developing new deduction techniques with improvements in terms of expressiveness and efficiency. Our ultimate goal is to obtain highly reliable procedure decisions that are nevertheless competitive with respect to state-of-the-art provers. In particular, we aim at developing specific strategies for SMT tools to scale up to verification conditions (aka proof obligations) generated from industrial applications.

Such proof obligations have a particular structure: they encode the execution of the program and the properties the program should satisfy. The problems contain two sets of axioms: one, common to all proof obligations, formalizes the programming environment (e.g., the C memory model of Why/Caduceus consists of around 80 axioms) while the other refines some aspects of the program computation and it is thus specific to a particular set of proof obligations. In practice, just a few axioms are needed to discharge every proof obligation and the useless axioms degrade the performances of SMT solvers in an unacceptable manner.

In this paper, we report on two experiments whose aim is to eliminate as many useless axioms as possible by suitable selection strategies: the first one is generic but external to the provers, while the second is hard-coded in the Alt-Ergo theorem prover.

In order to improve Alt-Ergo's quantifier reasoning module, we also implemented a new matching algorithm based on code trees. The first experimental results show substantial performance improvements over the existing matching algorithm.

2 Static Simplification of Program Verification Conditions

Possible solutions to reduce the verification condition (VC) size and complexity are to optimize the memory model (e.g. by introducing separations of zones of pointers [5]), to improve the weakest precondition calculus [6] and to apply strategies for simplifying VCs [4, 3, 7]. Our first experiment [2] focuses on the latter. We suggest heuristics to select axioms to feed automated theorem provers (ATPs). Instead of blindly invoking ATPs with a large VC, we present reduction strategies that significantly prune their search space. The idea behind these strategies is quite natural: an axiom is relevant if a prover applies it successfully, i.e. without diverging, to establish the conclusion. Relevance criteria are computed by the combined traversal of two graphs representing symbol dependencies within axioms. In the graph of constants edges represent the conjoint presence of two constants in some ground axiom. In the graph of predicates arcs represent logical dependencies between predicates occuring in the same axiom.

In former work [1], selection was limited to ground hypotheses and comparison predicates were not taken into account. This led to unsatisfactory results, for instance when the conclusion is some equality between terms. The method described in [2] extends selection to context axioms, comparison predicates and hypotheses with quantifiers. We propose new heuristics that increase the number of automatically discharged VCs.

The new method consists of a strategy that reduces program verification conditions by selecting their relevant hypotheses. The relevance of an hypothesis is determined by the combination of a syntactic analysis and two graph traversals. The first graph is labeled by constants and the second one by the predicates in the axioms (see Fig. 1). The approach was applied on a benchmark arising in the Oslo [10] secure bootloader, an industrial application.

Among the 771 generated proof obligations for Oslo, 741 were directly discharged, without any axiom selection. Next, the approach developed in [1] increased the result to 752. Thanks to the new method in [2], 10 more proof obligations were automatically proved as summarized in Fig. 2 with the three provers Simplify, Alt-Ergo 0.8 and Yices 1.0.20 with a timeout of 10 seconds.

3 Dynamic hypothesis selection in Alt-Ergo

We experimented hypothesis selection directly in the Alt-Ergo theorem prover. The advantage of this approach is that this selection can be tightly coupled with the SAT-solver and the matching algorithm of Alt-Ergo. This allows to aggressively prune hypotheses and refine this pruning during the proof search.

Our approach relies on the following two main mechanisms:

- a graph-based dependency analysis using function symbols, predicates and polarities;
- a refinement of the interaction of the SAT-solver and matching procedures in Alt-Ergo.

$\mathsf{valid}(a) \rightarrow$
$(\forall i : \mathsf{int}\,.\, 0 \leq i \leq 1 \Rightarrow \mathsf{valid}(a, \mathit{shift}(\mathit{acc}(m_v, a), i)) \ \wedge \mathsf{valid\_acc}(m_{pPM})) \rightarrow$
$\mathsf{valid\_acc\_range}(m_v, 2) \rightarrow$
$\mathsf{separation1\_range}(m_v, 2) \rightarrow$
$\mathsf{valid\_acc}(m_v) \rightarrow$
$\mathsf{not\_assigns}(m_x, \mathit{upd}(m_x, \mathit{shift}(\mathit{acc}(m_v, a), 0), 2),$
$\quad \mathit{singleton}(\mathit{acc}(m_v, a)))$

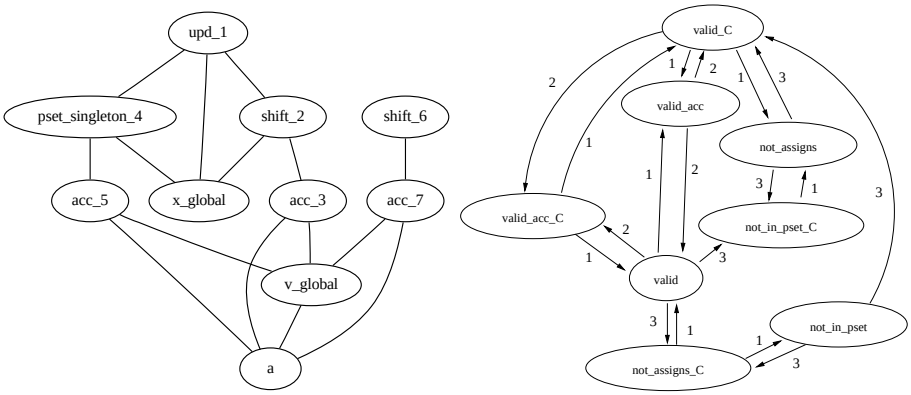


Figure 1: Example of hypothesis selection

Graph-based selection. Graphs are constructed such that nodes represent symbols (function or predicates) and edges are hypotheses relating symbols. The following Figure shows an excerpt of such a graph.

Edges link symbols that are consumed (negative polarity) to symbols that are produced (positive polarity). Starting from the known facts (green nodes), our algorithm tries to find a path to the wanted results (red nodes). The maximal length of the considered paths can be configured by the user.

Heuristics for instantiations We designed several heuristics to guide the instantiation mechanism of Alt-Ergo:

- we added information in the term data-structure in order to distinguish terms that directly come from the goal from the others ones (those coming from the hypotheses or from the context); we propagate this information at run-time, in particular for terms that are produced by instantiating lemmas. This is similar to the graph-based selection presented above: it allows to guide the proof search by favoring instances that are directly related to the goal;

- we also added information about the size of formulas and terms on order to give priority to smaller terms or formulas when trying to generate new instances of the available lemmas: the proof search first considers small terms and then gradually adds bigger terms if needed;

- we handled predicate definitions in a specific manner: unfolding of such

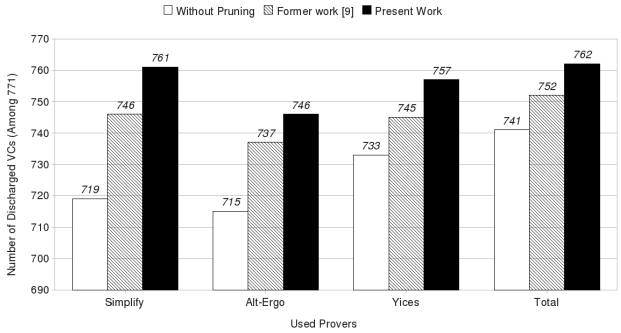
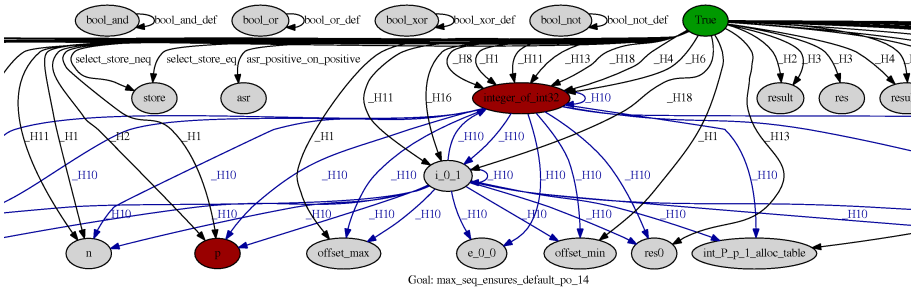


Figure 2: Result Comparison on Oslo Benchmark



definitions is done only when required and without considering the size of the body of the definitions.

4 New matching algorithm for Alt-Ergo

Sylvain Conchon and Jean-Christophe Filliâtre supervised the undergraduate internship of Arthur Milchior [8] from June to August 2009. Arthur improved a matching algorithm from L. de Moura and N. Bjørner [9] and implemented it in the Alt-Ergo theorem prover [8]. More precisely, the improvement consisted in handling types in the matching compilation to take polymorphism into account.

In order to check the efficiency of this new matching algorithm, we benchmarked Alt-Ergo on the Why test suite: 1792 verification conditions automatically generated by the Why tool from 30 programs. The first experimental results showed a 30% speed-up with respect to the current matching implementation.

5 Conclusion

We have presented two new strategies to select relevant hypotheses in formulae coming from program verification. The first one combines two separate dependency analyses based on graph computation and graph traversal, and uses some heuristics to analyse the graphs with a sufficient granularity. The relevance of

this approach has been demonstrated with a benchmark issued from a real industrial code. The second one has been directly implemented in the Alt-Ergo theorem prover. It is based on two mechanisms: a graph-based dependency analysis and a refined strategy for the creation of new instances.

In a near future we plan to apply these strategies to case studies coming from our industrial partners (Systerel and CEA). We also want to look at a proof search algorithm that would combine backward and forward reasoning in Alt-Ergo.

References

[1] J.-F. Couchot and T. Hubert. A Graph-based Strategy for the Selection of Hypotheses. In *FTP 2007 - International Workshop on First-Order Theorem Proving*, Liverpool, UK, Sept. 2007.

[2] J.-F. Couchot, A. Giorgetti and N. Stouls. Graph Based Reduction of Program Verification Conditions. In *AFM'09, Automated Formal Methods*, Grenoble, France, 2009.

[3] E. Denney, B. Fischer, and J. Schumann. An empirical evaluation of automated theorem provers in software certification. *International Journal on Artificial Intelligence Tools*, 15(1):81–108, 2006.

[4] E. P. Gribomont. Simplification of boolean verification conditions. *Theoretical Computer Science*, 239(1):165–185, 2000.

[5] T. Hubert and C. Marché. Separation analysis for deductive verification. In *Heap Analysis and Verification (HAV'07)*, Braga, Portugal, Mar. 2007.

[6] K. R. M. Leino. Efficient weakest preconditions. *Information Processing Letters*, 93(6):281–288, 2005.

[7] J. Meng and L. Paulson. Lightweight relevance filtering for machine-generated resolution problems. In *ESCoR: Empirically Successful Computerized Reasoning*, 2006.

[8] A. Milchior. Algorithme de matching, modulo égalité, incrémental, typé et persistant. *Undergraduate Internship, École Normale Supérieure, Paris*, 2009.

[9] L. de Moura and N. Bjørner. Efficient E-Matching for SMT Solvers. In Proceedings of the 21st international conference on Automated Deduction (CADE-21), Bremen, Germany, 2007.

[10] Bernhard Kauer. OSLO: Improving the security of Trusted Computing. In *16th USENIX Security Symposium, August 6-10, 2007, Boston, MA, USA*, 2007.