ARGO-LIB: A Generic Platform for Decision Procedures

Filip Marić[1] and Predrag Janičić[2]

[1] e-mail: filip@matf.bg.ac.yu
[2] e-mail: janicic@matf.bg.ac.yu
Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11 000 Belgrade, Serbia

Abstract. ARGO-LIB is a C++ library that provides support for using decision procedures and for schemes for combining and augmenting decision procedures. This platform follows the SMT-LIB initiative which aims at establishing a library of benchmarks for satisfiability modulo theories. The platform can be easily integrated into other systems. It also enables comparison and unifying of different approaches, evaluation of new techniques and, hopefully, can help in advancing the field.

1 Introduction

The role of decision procedures is very important in theorem proving, model checking of real-time systems, symbolic simulation, etc. Decision procedures can reduce the search space of heuristic components of such systems and increase their abilities. Usually, decision procedures have to be combined, to communicate with heuristic components, or some additional hypotheses have to be invoked. There are several influential approaches for using decision procedures, some of them focusing on combining decision procedures [8, 12, 1] and some focusing on augmenting decision procedures [2, 7]. We believe there is a need for an efficient generic platform for using decision procedures. Such a platform should provide support for a range of decision procedures (for a range of theories) and also for different techniques for using decision procedures. ARGO-LIB platform[1] is made motivated by the following requirements:

- it provides a flexible and efficient implementation of required primitives and also support for easily implementing higher-level procedures and different approaches for using decision procedures;
- it is implemented in the standard C++ programming language; the gains are in efficiency and in taking the full control over the proving process; the standard, plain C++ implementation is portable to different operating systems and there is no need for some specific operating system or some specific compiler; this also encourages and promotes sharing of code between different groups and enables converging implementation efforts in one direction;

[1] The web page for ARGO-LIB is at: www.matf.bg.ac.yu/~janicic/argo. The name ARGO comes from *Automated Reasoning GrOup* and from $A\rho\gamma\omega$, the name of the galley of argonauts (this galley was very light and very fast).

– it is "light" (i.e., small in size and stand-alone, requires only any standard C++ compiler) and fast (capable of coping with complex real-world problems using the existing or new approaches);
– its architecture is flexible and modular; any user can easily extend it by new functionalities or reimplement some modules of the system;
– it can work stand-alone but can also be integrated into some other tool (e.g., a theorem prover, constraint solver, model checking system etc.);
– it enables comparison on an equal footing between schemes for using decision procedures (especially when the worst-case complexity can be misleading);
– it is publicly available and it has simple, documented interface to all functionalities of the platform;
– it supports a library of relevant conjectures and a support for benchmarking (here we have in mind motivations, ideas, and standards promoted by the SMT-LIB initiative [10]); this enables exchange of problems, results, ideas, and techniques between different groups.

Although some of the above requirements might seem conflicting, we do believe that we have built a system that meet them in a high degree. Indeed, instead of making a new programming language and/or a system that interprets proof methods and that takes care about "protecting" its user, we opt for C++ and give the user more freedom and responsibility. Namely, it is sensible to assume that the user interested in using efficient decision procedures is familiar with C++ and for him/her it is simpler to install and to use some C++ compiler then some specific theorem proving platform. Therefore, and also because of efficient code, we believe that C++ is one of the best (or the best) options for a generic platform for decision procedures which aims at *realistic* wider use (both in academia and in industry).

2 Background

Concerning the background logic (first order classical logic with equality), underlying theories, description of theories, and format, ARGO-LIB follows the SMT-LIB initiative [10]. The main goal of the SMT-LIB initiative, supported by a growing number of researchers world-wide is to produce a library of benchmarks for satisfiability modulo theories and all required standards and notational conventions. Such a library will facilitate the evaluation and the comparison of different approaches for using decision procedures and advance the state of the art in the field. The progress that has been made so far supports these expectations.

In terms of logical organisation and organisation of methods, ARGO-LIB follows the proof-planning paradigm [3] and the GS framework [6]. The GS framework for using decision procedures is built from a set of methods some of which are abstraction, entailment, congruence closure, and "lemma invoking". There are generic and theory-specific methods. Some methods use decision procedures as black boxes, while some also use functionalities like elimination of variables. The GS framework is flexible and general enough to cover a range of schemes for both combining and augmenting decision procedures, including Nelson and Oppen's [8], Shostak's [12], and the approach used in the TECTON system [7].

3 Architecture

Representation of expressions ARGO-LIB contains a class hierarchy for representing first-order terms and formulae, with a wide set of low-level functions for manipulating them. Expressions are represented in a tree-like structure based on the Composite design pattern.

Unification and rewriting ARGO-LIB provides support for first order unification. The unification algorithm is encapsulated in the class Unification, so it can be easily altered. A generic rewriting mechanism (including reduction ordering) for terms and formulae is also supported. A set of rewrite rules can be provided directly in the code or it can be read from an external file.

Theories Each (decidable) theory inherits the base class Theory and is characterized by: *Signature*, *Decision procedure*, *Simplification procedure* (a function that performs theory-specific simplification of a given formula). At the moment, there is support for *PRA* (Presburger Rational Arithmetic), *FOLeq* (universally quantified fragment of equality), and *LISTS* (theory of lists).

Goals A goal is an object of the class Goal and is represented by: *List of underlying theories*, *Conjecture* (a first order logic formula over the combination of given theories), *Status* of the conjecture (satisfiable, valid, unsatisfiable, invalid), *Extra signature* (the list of uninterpreted symbols). A number of iterator classes (for iterating over the subgoals) is also provided. For example, the class GoalTheoriesIterator is used to iterate through the subgoals of a goal whose conjecture is a conjunction of (abstracted) literals defined over several theories — a subgoal for each theory is created and its conjecture consists of all literals of the initial goal that are defined over that theory.

Methods Each method inherits the abstract base class Method and transforms a given goal. A method has the following slots: *Name*, *Preconditions* (checks if the method is applicable to a given goal), *Effect* (a function that describes the effect of the method to a given goal), *Postconditions* (conditions that have to hold after the method application), *Parameters* (additional parameters, e.g., a variable to be eliminated). In ARGO-LIB, the methods don't have tactics attached and do not produce object-level proofs. A list of methods is constantly being expanded. Existing methods are divided in several groups: *general purpose methods* (e.g., Rewriting, Prenex, DNF, CNF, Skolemization, Negation, FOLSimplification), *theory specific methods* (e.g., NelsonOppenCCC, FourierMotzkinElimination, ListsSimplification), *combination of theories* (e.g., Abstraction, Partitioning, DeduceEqualityAndReplace, Simplification, UnsatModuloTheories), *combination schemes* (e.g., NelsonOppenCombinationScheme, TectonScheme).

Input format and parsing The native input format for ARGO-LIB is the SMT-LIB format (for benchmarks). Description of theories and their signatures are also read from the files in the (slightly extended) SMT-LIB format.

Output generating A number of classes (e.g., LaTeXOutput for LaTeX output, SMTOutput for SMT-LIB output, MathMLOutput for XML output) provide support for generating output in a readable and easy to understand format. The level and form of output information can be controlled.

4 Samples and Results

The example below shows the C++ code for Nelson-Oppen's scheme [8]. The procedure is entirely based on the use of the ARGO-LIB methods. Notice how the code faithfully reflects logical, high-level description of the procedure. Further below we show a part of one ARGO-LIB output. We have tested ARGO-LIB on a number of examples and the results are very good. For instance, CPU times are an order of magnitude less then for the PROLOG implementation reported in [6]. For instance, proving that $\forall x \forall y(x \leq y \wedge y \leq x + car(cons(0, x)) \wedge p(h(x) - h(y)) \Rightarrow p(0))$ is valid in combination of three theory takes 0.03s (on PC 700MHz).

```
void NelsonOppenCombinationScheme::Effect(Goal& g) {
    negation.ApplyTo(g);
    prenex.ApplyTo(g);
    skolemization.ApplyTo(g);
    dnf.ApplyTo(g);
    GoalDisjunctsIterator gdi(g);
    for (gdi.First(); !gdi.IsDone(); gdi.Next())
    {   Goal& current_disjunct=gdi.GetCurrent();
        abstraction_and_partitioning.ApplyTo(current_disjunct);
        while(1)
        {   simplify.ApplyTo(current_disjunct);
            unsat_modulo_theories.ApplyTo(current_disjunct);
            E_BOOL is_trivially_met=current_disjunct.IsTriviallyMet();
            if (is_trivially_met==TRUE)
            {  current_disjunct.SetTrue(); break;  }
            else if (is_trivially_met==FALSE)
            {  current_disjunct.SetFalse(); g.SetFalse(); return; }
            if (!deduce_equality_and_replace.ApplyTo(current_disjunct))
            {  current_disjunct.SetFalse(); g.SetFalse(); return; }
        }
    }
    g.SetTrue();
}
```

1. $(c_2 = h(c_3)) \wedge (c_1 + -2 * c_2 = 0) \wedge \neg(h(c_1 + -1 * c_2) = h(h(c_3)))$
   is *unsatisfiable* in the theory $\langle$PRA, FOL=$\rangle$
   if and only if (by the method *AbstractionAndPartitioning*)
2. $((c_1 + -2 * c_2 = 0) \wedge (c_1 + -1 * c_2 = c_4)) \wedge ((c_2 = h(c_3)) \wedge \neg(h(c_4) = h(h(c_3))))$
   is *unsatisfiable* in the theory $\langle$PRA, FOL=$\rangle$
   if and only if (by the method *Deduce Equality and Replace* (c_2=c_4))
3. $((c_1 - 2 * c_4 = 0) \wedge (c_1 - c_4 = c_4)) \wedge ((c_4 = h(c_3)) \wedge \neg(h(c_4) = h(h(c_3))))$
   is *unsatisfiable* in the theory $\langle$PRA, FOL=$\rangle$
   if and only if (by the method *Unsat Modulo Theory (FOL=)*)
4. $\top$ The goal has been met

5 Related Work

The work presented in this paper is related to the long line of results and systems for using decision procedures [8, 12, 1, 2, 7]. Especially, the ARGO-LIB builds on the GS framework [6]. Also, our work is related to several recent systems — systems being developed with similar goals (light and efficient support for decision procedures), including *haRVey* [9], *CVC Lite* [4], *TSAT++* [14], *ICS* [5].

6 Conclusions and Future Work

In this paper we presented ARGO-LIB — a C++ library that provides support for using a range of decision procedures and mechanisms for their combination and augmentation. The library builds upon ideas from the GS framework and the SMT-LIB initiative. We hope that ARGO-LIB will promote exchange of ideas, techniques, benchmarks and code and so help advance in the field. For further work we are planning to extend ARGO-LIB by new decision procedures and also to further improve some low-level modules (e.g., unification). We are planning to test ARGO-LIB on wide sets of real-world problems and to compare it to the rival systems. We will also look for its applications in systems of industrial strength.

References

1. C. W. Barrett, D. L. Dill, and Aaron Stump. A Framework for Cooperating Decision Procedures. *CADE-17*, LNAI 1831. Springer, 2000.
2. R. S. Boyer and J S. Moore. Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic. *Machine Intelligence 11*, 1988.
3. A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. *CADE-9*, Springer, 1988.
4. CVC Lite. on-line at: http://verify.stanford.edu/CVCL/.
5. ICS. on-line at: http://www.icansolve.com/.
6. P. Janičić and A. Bundy. A General Setting for the Flexible Combining and Augmenting Decision Procedures. *Journal of Automated Reasoning*, 28(3), 2002.
7. D. Kapur and M. Subramaniam. Using an induction prover for verifying arithmetic circuits. *Software Tools for Technology Transfer*, 3(1), 2000.
8. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2), 1979.
9. S. Ranise and D. Deharbe. Light-weight theorem proving for debugging and verifying units of code. *SEFM-03)*. IEEE Computer Society Press, 2003.
10. S. Ranise and C. Tinelli. The SMT-LIB Format: An Initial Proposal. 2003. on-line at: http://goedel.cs.uiowa.edu/smt-lib/.
11. H. Rueß and N. Shankar. Deconstructing Shostak. In *Proceedings of the Conference on Logic in Computer Science (LICS)*, 2001.
12. R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1), January 1984.
13. Aaron Stump, Arumugam Deivanayagam, Spencer Kathol, Dylan Lingelbach, and Daniel Schobel. Rogue deicision procedures. *Workshop PDPAR*. 2003,
14. TSAT++. on-line at: http://www.mrg.dist.unige.it/Tsat.