CSIsat: Interpolation for LA+EUF*
Tool Paper

Dirk Beyer[1], Damien Zufferey[2], and Rupak Majumdar[3]

[1] Simon Fraser University, BC, Canada
[2] EPFL, Switzerland
[3] UCLA, CA, USA

Abstract. We present CSIsat, an interpolating decision procedure for the quantifier-free theory of rational linear arithmetic and equality with uninterpreted function symbols. Our implementation combines the efficiency of linear programming for solving the arithmetic part with the efficiency of a SAT solver to reason about the boolean structure. We evaluate the efficiency of our tool on benchmarks from software verification. Binaries and the source code of CSIsat are publicly available as free software.

1 Overview

The *Craig interpolant* for a pair (ϕ_1, ϕ_2) of formulas such that $\phi_1 \wedge \phi_2$ is not satisfiable, is a formula ψ such that ϕ_1 implies ψ, the conjunction $\psi \wedge \phi_2$ is not satisfiable, and ψ is over symbols that are common to ϕ_1 and ϕ_2 [4]. Craig interpolants have been applied successfully in formal verification and logic synthesis. For example, several software verification tools use Craig interpolants derived from infeasible counterexamples to refine their abstractions.

An *interpolating decision procedure* extends a decision procedure in the following way: it takes as input a pair (ϕ_1, ϕ_2) of formulas and has two possible outcomes: the procedure returns (1) with the answer SAT, if the conjunction $\phi_1 \wedge \phi_2$ is satisfiable, or otherwise (2) with a formula ψ that is a Craig interpolant for (ϕ_1, ϕ_2). CSIsat [1] is a new tool that implements an interpolating decision procedure for boolean combinations of linear-arithmetic expressions and equality with uninterpreted function symbols (LA+EUF).

Availability. The source code, executables, and all benchmarks for CSIsat are available online at `http://www.cs.sfu.ca/~dbeyer/CSIsat/`. The tool is free software, released under the GPLv3 license. CSIsat is the first open-source interpolating decision procedure available to verification researchers. We hope that other researchers can integrate new interpolating decision procedures into CSIsat and that developers find it easy to integrate the tool into more applications.

* Dirk Beyer and Damien Zufferey were supported in part by the Canadian NSERC grant RGPIN 341819-07; Rupak Majumdar was supported in part by the NSF grants CCF 0546170 and CCF 0720882.

[1] Available at `http://www.cs.sfu.ca/~dbeyer/CSIsat/`

A. Gupta and S. Malik (Eds.): CAV 2008, LNCS 5123, pp. 304–308, 2008.

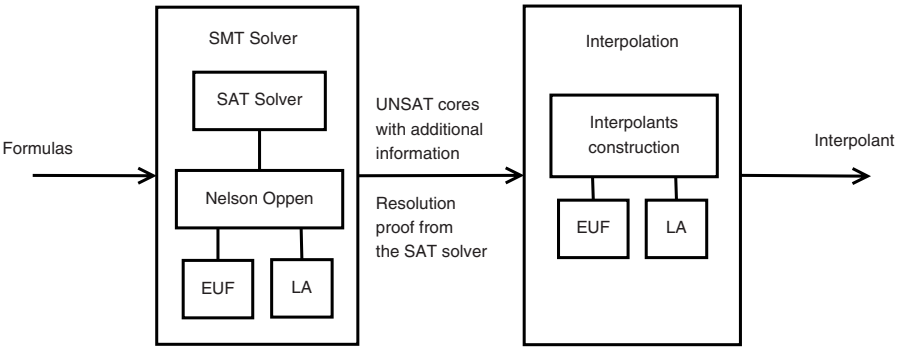

Fig. 1. Architecture of CSIsat

Related Tools. So far there are two published interpolation tools: Foci and CLPprover. McMillan's tool Foci [2] is an interpolation procedure for boolean combinations of linear-arithmetic expressions and equality with uninterpreted function symbols [6]. The tool is implemented as a proof-based theorem prover. Rybalchenko's tool CLPprover [3] is an interpolation procedure for *conjunctions* of linear-arithmetic constraints and equality with uninterpreted function symbols [9]. The tool is based on linear-constraint solving and implemented on top of the CLP($\mathbb{Q}$,$\mathbb{R}$) library [5] for SICStus Prolog.

These two existing tools have different advantages over each other: CLPprover takes advantage of linear-constraint solving and can provide an efficient solution for conjunctions of linear-arithmetic expressions, and can constrain interpolants to be only over particular variables, if possible. Foci, on the other hand, handles boolean combinations efficiently. CSIsat combines the advantages of both approaches, and uses efficient SMT algorithms to provide a fast interpolation procedure. Our experimental evaluation provides evidence of good performance. [4]

2 Architecture and Algorithm

Figure 1 illustrates the architecture of CSIsat. Our goal is to provide a tool for computing interpolants for boolean combinations of (rational) linear-arithmetic expressions (LA) and equality with uninterpreted function symbols (EUF). Interpolants for pure conjunctions of linear-arithmetic constraints can be efficiently computed using linear programming, and therefore, CSIsat uses the algorithm

[2] Available at `http://www.kenmcmil.com/foci.html`

[3] Available at `http://www.mpi-sws.mpg.de/~rybal/clp-prover/`

[4] The original motive for our work was a very practical one: Until now, we had two interpolation tools integrated in Blast: Foci and CLPprover. To verify different programs we had to use different command-line options: -foci by default, and -clp for programs that require to track linear-arithmetic expressions.

of Rybalchenko and Sofronie-Stokkermans to compute interpolants for such formulas [9].

The constraint-based algorithm cannot directly handle formulas that are not convex in their geometrical interpretation. To solve this problem, Rybalchenko and Sofronie-Stokkermans propose to convert both formulas ϕ_1, ϕ_2 to disjunctive normal form (DNF), perform multiple queries to the CLP-based algorithm and construct the final interpolant from the results of these queries [9]. Unfortunately, the DNF conversion can often blow up in practice.

As a solution to this problem, we chose a two-step approach. For the first step we use an SMT solver that integrates SAT with Nelson-Oppen style theory reasoning [7, 2, 8]. If the conjunction of the formulas is satisfiable, the tool stops with answer SAT and returns a satisfiable subformula that implies the conjunction of the two input formulas. If the conjunction is not satisfiable, CSIsat collects the unsatisfiable core and computes the interpolants from this. For this second step, we annotate the unsatisfiable core with additional information to avoid overhead. This information comprises the equalities deduced by theory-specific reasoning, and is used to compute partial interpolants. In addition, the resolution proof from the SAT solver is passed to the interpolation step. We use McMillan's approach to construct interpolants for the EUF part [6]; the rules were adapted to a graph-based framework. We construct interpolants for the linear-arithmetic specific part using the constraint-interpolation technique [9], and combine the interpolants using the technique of Yorsh et al. [10].

The interface to our tool is taken from Foci, i.e., CSIsat uses the same syntax for the input formulas, and the same output syntax for the interpolants, such that we can easily substitute one tool for the other. Our implementation is based on two domain-specific components. For the linear-constraint solving part, we use the GNU Linear Programming Kit (GLPK) [5]. Our SMT algorithm is based on an integrated SAT solver component. We have successfully experimented with substituting PicoSAT [6] [1] for our own SAT solver. The linear-programming component and the SAT solver component are both integrated through a wrapper interface, which can easily be adapted to other linear-programming or SAT solver components.

3 Performance Results

We show that CSIsat is competitive by comparing all three publicly available interpolation tools on some motivating examples from the software model checker Blast. Our experiments indicate that CSIsat can efficiently find interpolants.

All experiments were performed on a GNU/Linux x86_64 machine with an Intel Core 2 Duo processor and 2 GB RAM. We limited the processor speed to 1 GHz, in order to emphasize the difference. We report only the consumed User CPU Time, in order to reduce the bias from input/output operations and overhead for process setup. For all software components in our experiments,

[5] Available at `http://www.gnu.org/software/glpk/`

[6] Available at `http://fmv.jku.at/picosat/`

Table 1. Performance evaluation on Blast verification benchmarks

Program	#queries	Foci	CLPprover	CSIsat
kbfiltr	64	0.28 s	0.14 s	0.10 s
floppy	235	1.17 s	1.55 s	0.55 s
diskperf	119	0.56 s	0.61 s	0.23 s
cdaudio	130	0.60 s	0.70 s	0.26 s
ssh	6881	29 s	—	17 s
alias_swap.c	8 (908)	0.07 s	13.20 s	0.06 s

we used the latest publicly available versions, as of April 21, 2008: CSIsat 1.1; CLPprover 0.22; Foci 2003; GLPK 4.28; PicoSAT 632.

Table 1 reports the run times of the three tools on interpolation queries that occur during verification processes of different programs. The first column identifies each program that was verified by Blast. The first four programs are MS Windows device drivers, `ssh` consists of several files from the SSH software that were instrumented for verifying different properties, and `alias_swap.c` is from Blast's regression test base. During each verification run, we dumped all interpolation queries to files. Then we ran the interpolation procedures once again only on the queries, and the time in the table is the sum of the run times over all queries that were dumped for a program. The `ssh` experiment consisted of 19 verification tasks (program code and property), each resulting in about 350 interpolation queries. The row in the table reports the sum of the run times over all 19 verification tasks. The `ssh` interpolation queries contain a high number of subformulas of the form $a \neq b$. CLPprover does not support disjunctions, and transforming each such subformula to $a < b \vee a > b$, and the resulting overall formula into DNF, resulted in an intractable number of conjunctive queries. The example `alias_swap.c` requires interpolants for formulas with disjunctions, because it uses pointer aliases. In this case, we did the transformation to DNF in order to feed CLPprover. These last two examples demonstrate the importance of efficient handling of boolean combinations.

Acknowledgments. We thank Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani for interesting discussions relating to interpolation, and a pointer to their recent paper [3].

References

1. Biere, A.: PicoSAT essentials. JSAT (submitted, 2008)
2. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T.A., Ranise, S., Rossum, P.v., Sebastiani, R.: Efficient satisfiability modulo theories via delayed theory combination. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 335–349. Springer, Heidelberg (2005)
3. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient interpolant generation in satisfiability modulo theories. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 397–412. Springer, Heidelberg (2008)

4. Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symb. Log. 22(3), 250–268 (1957)
5. Holzbaur, C.: OFAI clp(q,r) Manual, Edition 1.3.3. Austrian Research Institute for Artificial Intelligence, Vienna, TR-95-09 (1995)
6. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. 345(1), 101–121 (2005)
7. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM 27(2), 356–364 (1980)
8. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). J. ACM 53(6), 937–977 (2006)
9. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint Solving for Interpolation. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 346–362. Springer, Heidelberg (2007)
10. Yorsh, G., Musuvathi, M.: A combination method for generating interpolants. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 353–368. Springer, Heidelberg (2005)