SatAbs: A Bit-Precise Verifier for C Programs*
(Competition Contribution)

Gérard Basler, Alastair Donaldson[1], Alexander Kaiser[2], Daniel Kroening[2], Michael Tautschnig[2], and Thomas Wahl[3]

[1] Imperial College, London, United Kingdom
[2] University of Oxford, United Kingdom
[3] Northeastern University, Boston, United States

Abstract. SatAbs is a bit-precise software model checker for ANSI-C programs. It implements sound predicate-abstraction based algorithms for both sequential and concurrent software.

1 Verification Approach

SatAbs [7] is a verifier for C programs that uses counterexample-guided abstraction refinement [8] (Fig. 1), based on predicate abstraction [12], as pioneered by SLAM [2]. By interpreting variables of the C program as bit-vectors, efficient SAT procedures are used for abstraction and simulation [6]. This renders the theorem prover calls that are made during abstraction decidable, and enables bit-precise verification, which is essential when analysing system-level software.

In [10] the first sound and symmetry-aware predicate abstraction based approach towards model checking multi-threaded programs was presented. These results have now been integrated with SatAbs, allowing scalable verification of concurrent C programs comprised of replicated threads.

Efficient symmetry-aware predicate abstraction requires amendments in all four key components of Figure 1: (1) the Boolean program computed as abstraction will use *passive predicates and broadcasts* [10]; (2) the underlying model checker for Boolean programs must be able to make use of symmetry and support passive predicates, which presently only Boom [3] does; (3) as adding new predicates makes Boolean program model checking more expensive, we primarily rely on *transition refinement* (cf. [1]) – symmetry-aware analysis requires a particular variant that handles both active and passive threads; (4) adding new predicates in case of replicated threads requires extra care (cf. [10]).

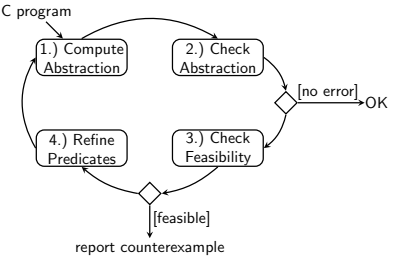

Fig. 1. Key components of SatAbs

* This research is supported by EPSRC projects EP/G026254/1, EP/G051100/1 and EP/H017585/1 and ERC project 280053.

C. Flanagan and B. König (Eds.): TACAS 2012, LNCS 7214, pp. 552–555, 2012.

2 Architecture

SatAbs shares large portions of the underlying C++ framework with CBMC [5], including the ANSI-C front end, the internal representation as GOTO programs, and interfaces to decision procedures. The competition candidate is linked with MiniSat 2.2.0 [11] as SAT solver.

Boolean Program Model Checking. Boolean program model checking is typically the bottleneck for a CEGAR-based verifier. SatAbs treats the Boolean program model checker as black box, and can be configured to use any suitable tool. For the competition, we use either Boom [4] together with a wrapper to analyse concurrent programs with as few threads as necessary, or Cadence SMV.

Refinement Strategy. SatAbs starts verification with a coarse abstraction, cheaply computed over predicates derived from assertions appearing in the program. The abstraction is refined in response to spurious counterexamples according to the following strategy: first, a spurious counterexample is checked for spurious transitions. If any exist, they are refined away using the technique of Das and Dill [9], following the approach of SLAM [1]. If no individual transition is spurious, weakest precondition calculations are used to derive new predicates from the counterexample, which are used to compute a more precise abstraction.

3 Strengths and Weaknesses

SatAbs supports all categories, including "Concurrency". In the presence of replicated concurrent programs, SatAbs exploits symmetry [13] to curb state explosion. Although the tool can also be applied to asymmetric concurrent programs, no scalability is expected for this case. As the focus on symmetric threads is not reflected in the benchmarks, SatAbs timed out on most of the benchmarks in the category "Concurrency". In the category "HeapManipulation", SatAbs failed because of a bug in the counterexample analysis; this has been fixed and in future we expect positive results there as well.

Overall, SatAbs proved to be reliable: bit-precise reasoning paired with some degree of maturity made SatAbs return only a single wrong result, which was due to bugs that have been fixed in the meantime. Yet we are aware of several limitations and weaknesses, which concern both sequential and concurrent code. Current technical limitations of predicate discovery may lead to SatAbs reporting "refinement failures". Furthermore overall efficiency and performance require closer inspection to reduce the number of timeouts that SatAbs had in the competition.

4 Tool Setup

SatAbs is hosted at http://www.cprover.org/satabs/ and is available both in binary form for popular platforms and as source code under a 4-clause BSD license. A C preprocessor is required (as provided by GCC on Unix-like platforms or

Visual Studio on Microsoft Windows). The model checkers Boom and Cadence SMV were used in the competition – SMV must be downloaded separately.[1]

The following command-line options were used for the competition, depending on category: 1) `--modelchecker boom`: Select Boom as model checker; without this option, Cadence SMV is used as default. 2) `--full-inlining`: Inline all functions. This is required for proper operation when using Boom. 3) `--error-label ERROR`: Instead of searching for violated assertions, prove (un)reachability of the label "ERROR" as specified in the competition rules. 4) `--32`: Select the basic bit-width of the architecture; by default, the bit-width of the execution platform is assumed, but some categories were designated to contain 32-bit benchmarks. 5) `--concurrency`: Enable use of passive threads and broadcast assignments, as described above. 6) `--max-threads 5`: Passed to Boom: only analyse executions involving no more than 5 concurrently running threads. 7) `--iterations 500`: Sets the upper bound on the number of CEGAR iterations to 500.

For categories "ControlFlowInteger" and "SystemC" we used SMV as model checker, i.e., option 1) was not given. Options 5) and 6) were only used for the category "Concurrency".

References

1. Ball, T., Cook, B., Das, S., Rajamani, S.K.: Refining Approximations in Software Predicate Abstraction. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 388–403. Springer, Heidelberg (2004)
2. Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: PLDI, pp. 203–213 (2001)
3. Basler, G., Hague, M., Kroening, D., Ong, C.-H.L., Wahl, T., Zhao, H.: Boom: Taking Boolean Program Model Checking One Step Further. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 145–149. Springer, Heidelberg (2010)
4. Basler, G., Mazzucchi, M., Wahl, T., Kroening, D.: Context-aware counter abstraction. Formal Methods in System Design 36(3), 223–245 (2010)
5. Clarke, E., Kroning, D., Lerda, F.: A Tool for Checking ANSI-C Programs. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 168–176. Springer, Heidelberg (2004)
6. Clarke, E., Kroening, D., Sharygina, N., Yorav, K.: Predicate abstraction of ANSI–C programs using SAT. Formal Methods in System Design (FMSD) 25, 105–127 (2004)
7. Clarke, E., Kroning, D., Sharygina, N., Yorav, K.: SATABS: SAT-Based Predicate Abstraction for ANSI-C. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 570–574. Springer, Heidelberg (2005)
8. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
9. Das, S., Dill, D.L.: Successive approximation of abstract transition relations. In: LICS (2001)

[1] Available at http://www.kenmcmil.com/

10. Donaldson, A., Kaiser, A., Kroening, D., Wahl, T.: Symmetry-Aware Predicate Abstraction for Shared-Variable Concurrent Programs. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 356–371. Springer, Heidelberg (2011)
11. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
12. Graf, S., Saïdi, H.: Construction of Abstract State Graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
13. Wahl, T., Donaldson, A.F.: Replication and abstraction: Symmetry in automated formal verification. Symmetry 2(2), 799–847 (2010)