SLAB: A Certifying Model Checker for Infinite-State Concurrent Systems *

Klaus Dräger[1], Andrey Kupriyanov[1], Bernd Finkbeiner[1], Heike Wehrheim[2]

[1] Universität des Saarlandes, Saarbrücken, Germany
[2] Universität Paderborn, Germany

Abstract. Systems and protocols combining concurrency and infinite state space occur quite often in practice, but are very difficult to verify automatically. At the same time, if the system is correct, it is desirable for a verifier to obtain not a simple ”yes” answer, but some independently checkable certificate of correctness. We present SLAB — the first certifying model checker for infinite-state concurrent systems. The tool uses a procedure that interleaves automatic abstraction refinement using Craig interpolation with slicing, which removes irrelevant states and transitions from the abstraction. Given a transition system and a safety property to check, SLAB either finds a counterexample or produces a certificate of system correctness in the form of inductive verification diagram.

1 Slicing Abstractions

SLAB (for *sl*icing *ab*stractions) is an automatic certifying model checker that implements the *abstraction refinement* loop presented in [1]. It interleaves refinement steps with *slicing*, which tracks the dependencies between variables and transitions in a system and removes irrelevant parts.

SLAB maintains an explicit graph representation of the abstract model: each node represents a set of concrete states, identified by a set of predicates; each edge represents a set of concrete transitions, identified by their transition relations.

Starting with the initial abstraction, the abstract model is transformed by refinement and slicing steps until the system is proved correct or a concretizable error path is found.

A refinement step increases the precision of the abstraction by introducing a new predicate, which is obtained by Craig interpolation from the unsatisfiable formula corresponding to some spurious error path. To minimize the increase in the size of the graph, the new predicate is only applied to one specific node on the error path. This node is split into two copies, the labels of which now additionally contain the new predicate and its negation, respectively.

A slicing step reduces the size of the abstraction while maintaining all error paths. *Elimination rules* drop nodes and edges from the abstraction if they have

* This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS, www.avacs.org).

become unreachable or if their label has become unsatisfiable. *Simplification rules* remove constraints from transition relations that have become irrelevant and simplify the graph structure of the abstraction.

2 Certifying Model Checker

If SLAB proves a concurrent system correct, then it produces from the final abstraction an efficiently and independently checkable certificate of the correctness. Such a certificate is much more useful than the usual binary response "correct"/"incorrect" (see e.g. [6]): it provides higher *degree of confidence* in the results of the verification run; it can be employed in *automated theorem proving* by importing it into a theorem prover and composing with certificates from other subgoals into a single proof; finally, it can be used to obtain *proof-carrying code.*

SLAB produces certificates in the form of inductive verification diagrams [5]: directed graphs in which nodes n are labeled with state predicates φ_n, and edges — with sets of transition relations. They satisfy the following properties (where we use Φ for the disjunction over all φ_n):

- Every initial state is represented by some node, i.e., *init* implies Φ;
- If a state s is represented by a node and has an outgoing transition $s \xrightarrow{\tau} s'$, then s' is also represented by some node. Equivalently, for each transition relation τ, the Hoare triple $\{\Phi\}\tau\{\Phi\}$ must be valid.
- Every node label precludes the error condition, i.e. Φ implies $\neg error$.

Thus the disjunction of the certificate node labels forms an inductive invariant which ensures that an error can never occur, and checking the correctness of the certificate boils down to verifying the above conditions on Φ.

The edge labels provide an alternative set of simpler Hoare triples for the second condition: For each node n and transition τ, $\{\varphi_n\}\tau\{\bigvee_m \varphi_m\}$ must hold, where the disjunction is over all m with $n \xrightarrow{\tau} m$.

As an example, Fig. 2 shows the specification of a simplified ring-buffer for a double-ended queue, consisting of cells (represented by integer variables) which can be either free (0) or occupied (1). Starting with a single occupied cell x_1, we can toggle a cell's state if the states of its neighbors differ.

init	$x_1 = 1 \wedge x_2 = 0 \wedge \cdots \wedge x_n = 0$
error	$x_1 = 1 \wedge x_2 = 1 \wedge \cdots \wedge x_n = 1$
τ_1	$x_n + x_2 = 1 \wedge x_1' = 1 - x_1 \wedge \Delta(\{x_1\})$
τ_2	$x_1 + x_3 = 1 \wedge x_2' = 1 - x_2 \wedge \Delta(\{x_2\})$
$\vdots$	$\vdots$
τ_n	$x_1 + x_{n-1} = 1 \wedge x_n' = 1 - x_n \wedge \Delta(\{x_n\})$

Fig. 1. Initial condition, error condition, and transitions of the deque example. $\Delta(S)$ denotes the frame condition that all variables $x \notin S$ remain unchanged.

The certificate produced by SLAB for an instance with 5 cells is shown on Fig. 2. The inductivity of the diagram guarantees that no error state is reachable.

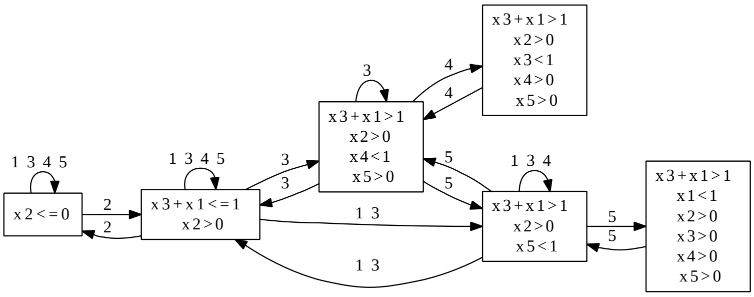

Fig. 2. A certificate of correctness for the deque example.

3 Results

SLAB is written in C++, and is available for the Linux platform. As an underlying mechanism for satisfiability checking and Craig interpolation, SLAB uses the MathSAT 4 SMT solver [2].

The model checker produces certificates both in graphical format as in Fig. 2 for visual inspection by the user, and in the SMT-LIB format, which can be checked by any of a large number of standard SMT solvers.

The user can customize several parameters of the abstraction refinement loop:

- The *initial abstraction*: The user can choose either a simple 4-state abstraction, based on the initial and error conditions, or a control flow graph.
- The *trace selection strategy*: The user can choose between random and deterministic selection of traces.
- The *node splitting strategy*: The user may allow several nodes to be split along any unsatisfiable trace of the abstraction.

Table 1 shows the performance of SLAB on a range of benchmarks. For comparison, we also give the running times of the *Abstraction Refinement Model Checker* ARMC [7], the *Berkeley Lazy Abstraction Software Verification Tool* BLAST [4] and the *New Symbolic Model Checker* NuSMV [3], where applicable. The benchmarks include a finite-state concurrent systems (Deque and Philosophers), an infinite-state discrete system (Bakery), and a real-time system (Fisher). BLAST and NuSMV are not applicable to the real-time system Fischer. Because NuSMV is able to verify only finite state systems, the running times for this tool are given for two cases: when all integer variables are bounded to 10 and 100 values.

	ARMC	BLAST	NuSMV		SLAB	
Benchmark	time	time	time (10)	time (100)	time	certificate size
Deque 5	1.81	0.55	0.03	5.64	0.06	6
Deque 10	776.33	2.32	0.05	13.89	0.18	11
Deque 15	timeout	6.40	0.08	22.71	0.40	16
Deque 20	timeout	13.41	0.14	36.08	0.69	20
Bakery 2	2.26	21.71	0.03	0.72	0.43	25
Bakery 3	33.44	134.72	0.04	6.44	1.45	35
Bakery 4	753.15	error	0.17	293.65	4.00	45
Bakery 5	timeout	879.71	0.33	timeout	10.26	55
Philosophers 3	125.82	15.02	0.24	7.82	0.76	11
Philosophers 4	timeout	92.04	0.89	25.16	3.05	26
Philosophers 5	timeout	658.80	4.36	554.86	11.50	57
Philosophers 6	timeout	timeout	9.24	timeout	42.57	120
Fischer 2	1.45	N/A	N/A	N/A	0.65	24
Fischer 3	48.68	N/A	N/A	N/A	9.16	170
Fischer 4	1842.85	N/A	N/A	N/A	122.77	1014

Table 1. Experimental results of SLAB, comparing its performance on a range of benchmarks to the tools ARMC, BLAST and NuSMV. Running times are given in seconds, with a timeout of 1 hour. All benchmarks were measured on AMD Opteron 2.6Ghz processors.

Availability. SLAB is available online at http://react.cs.uni-sb.de/slab, including documentation and the benchmarks used in this paper.

Acknowledgements. We would like to thank Alberto Griggio for fruitful discussions about the MathSAT 4 SMT solver.

References

1. I. Brückner, K. Dräger, B. Finkbeiner, and H. Wehrheim. Slicing abstractions. In F. Arbab and M. Sirjani, editors, *FSEN*, volume 4767 of *LNCS*, 2007.
2. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. TheMathSAT 4 SMT solver. In *CAV*, volume 5123 of *LNCS*, pages 299–303. Springer, 2008.
3. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource tool for symbolic model checking. In *CAV*, volume 2404 of *LNCS*, pages 241–268. Springer, 2002.
4. T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with BLAST. In *SPIN*, volume 2648 of *LNCS*. Springer, 2003.
5. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer, 1995.
6. K. Namjoshi. Certifying model checkers. In *CAV*, volume 2102 of *LNCS*, pages 2–13. Springer, 2001.
7. A. Podelski and A. Rybalchenko. ARMC: The logical choice for software model checking with abstraction refinement. In *PADL*, volume 4354 of *LNCS*, pages 245–259. Springer, 2007.