Bit-Precise LTL Model Checking

Petr Bauch
Faculty of Informatics, Masaryk University

FMCAD 2014 Student Forum

ParaDiSe Parallel & Distributed Systems Laboratory

Abstract

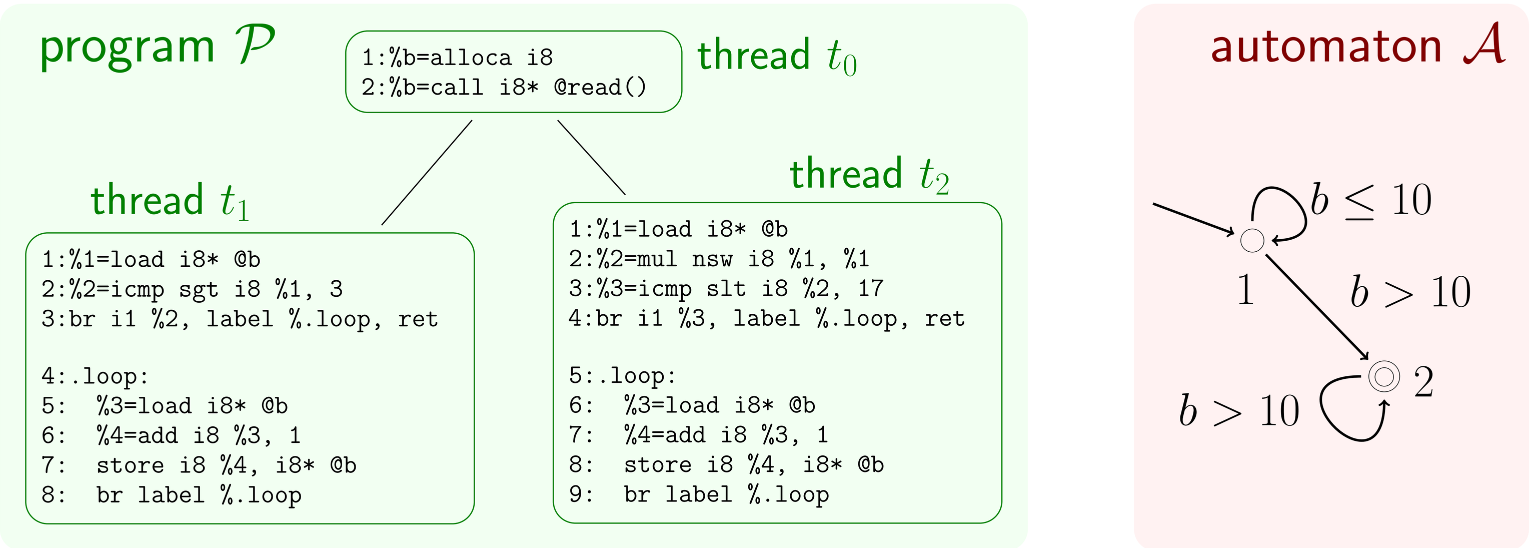
Complete verification of unmodified code is a challenging task, well-motivated by the costs of software debugging. In this work we rise to the challenge by proposing a model checking method that operates on unmodified parallel programs, specifically accepting LLVM bitcode as input. Apart from being complete, our method proves correctness of a program w.r.t. a temporal specification and is sound w.r.t. arithmetic overflows of integer variables. To overcome the limitations of classical model checking: state space explosion, state matching, etc. we further propound to reduce the model checking problem to a specific instance of the non-termination checking and lift the recently proposed *property directed reachability* to compute approximations of *recurrent sets*.

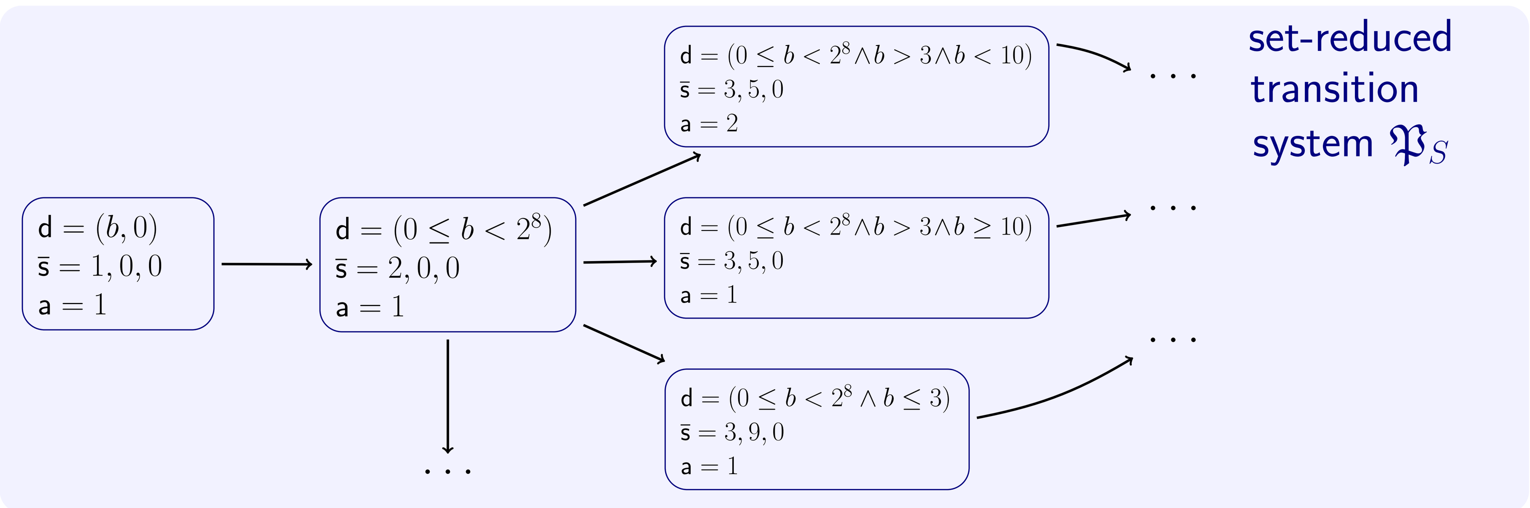
1 Present State

Our tool accepts LLVM bitcode as the program and an LTL formula as the specification where the atomic propositions are quantifier free *bit-vector* formulae over global variables. Since variable evaluations are also represented as bit-vector formulae, comparing two states while searching for a fair cycle within the system can be reduced to *satisfiability modulo theories* query. The major limitation of the present state arises from this query being quantified, thus increasing the complexity of individual state comparisons.

1.1 Work Flow

The figure below illustrate how control explicit—data symbolic model checking transforms the input pair (program $\mathcal{P}$, specification $\mathcal{A}$) to generate the set-reduced transition system $\mathfrak{P}_S$. The formulae representing variable evaluations are cumulatively collected along the paths in both the program and the specification automaton.





The set-based reduction using bit-vector formulae is not the only output of our tool SymDivine [1]. We can also generate the state space encoded with BDDs or generate the control flow graph (supporting other tools with access to parallel programs).

1.2 Experiments

We have evaluated SymDivine on examples translated from C programs. Apart from Erik Koskinen's tool (reducing model checking to termination analysis) we also compare with nuXmv (using SMT-based bounded model checking): two state-of-the-art model checkers.

Name	Property	Koskinen	nuXmv	SymDi
acqrel	$\mathcal{G}(a \Rightarrow \mathcal{F}\, b)$	14.18	0.31	28.25
apache	$\mathcal{G}(a \Rightarrow \mathcal{G}\,\mathcal{F}\, b)$	197.4	>60	47.50
fig8-2007	$\mathcal{G}(a \Rightarrow \mathcal{G}\,\mathcal{F}\, b)$	27.94	0.25	>60
pgarch	$\mathcal{G}\,\mathcal{F}\, a$	15.20	>60	0.83
win1	$\mathcal{G}(a \Rightarrow \mathcal{F}\, a)$	539.0	7.15	0.62
win3	$\mathcal{F}\,\mathcal{G}\, a$	15.75	0.04	0.18
1	$\mathcal{F}\,\mathcal{G}\, a$	11.0	2.43	0.7
12	$\mathcal{F}(a \wedge \neg \mathcal{G}\, a)$	3.9	0.41	0.23

References

[1] **LTL Model Checking of LLVM Bitcode with Symbolic Data** P. Bauch, V. Havel, J. Barnat, To appear in *Proc. of MEMICS*, 2014

[2] **An Incremental Approach to Model Checking Progress Properties** A. Bradley, F. Somenzi, Z. Hassan, Z. Yan, In *Proc. of FMCAD*, 2011

[3] **Proving Non-termination** A. Gupta, T. Henzinger, R. Majumdar, A. Rybalchenko, R. Xu, In *Proc. of POPL*, 2008

[4] **QF_BV Model Checking with Property Directed Reachability** T. Welp, A. Kuehlmann, In *Proc. of DATE*, 2013

2 Avoid Loop Unrolling: Idea

Apart from expensive state comparison the present approach is also limited by the fact that program loops may be exhaustively unrolled during verification. A recently proposed property guided reachability (PDR) [2] avoids unrolling loops while also allowing extensions for LTL and CTL model checking by proving unreachability among fair states. We propose an alternative extension via non-termination analysis [3], specifically to lift PDR to compute approximations of recurrent sets of evaluations pertaining to fair states.

Formalisation

Definition: By $Reach(l)$ we denote the set of evaluations at location l reachable from the initial location, i.e. $\forall \mathbf{x} \in Reach(l), (l_0, \mathbf{x}_0) \rightsquigarrow (l, \mathbf{x})$.
Definition: By $Recur(l)$ we denote a set of evaluations, such that $\forall \mathbf{x} \in Recur(l)$ there is a path from l to l that performs an identity on $\mathbf{x}$, i.e. $(l, \mathbf{x}) \rightsquigarrow (l, \mathbf{x})$.
Definition: Let X be a set of evaluations characterised by Φ, i.e. $\forall \mathbf{x}, \Phi(\mathbf{x}) \Leftrightarrow \mathbf{x} \in X$. Then by $\uparrow X$ we denote any *over-approximation* of X, i.e. $\forall \mathbf{x}, \Phi(\mathbf{x}) \Rightarrow \mathbf{x} \in \uparrow X$. And by $\downarrow X$ we denote any *under-approximation* of X, i.e. $\forall \mathbf{x}, \Phi(\mathbf{x}) \Leftarrow \mathbf{x} \in \downarrow X$.

Theorem: For a program $\mathcal{P}$ and a specification φ let F be the set of fair locations. Then

$$\exists l_F \in F, \downarrow Reach(l_F) \cap \downarrow Recur(l_F) \neq \emptyset \Rightarrow \mathcal{P} \not\models \varphi$$

and dually

$$\forall l_F \in F, \uparrow Reach(l_F) \cap \uparrow Recur(l_F) = \emptyset \Rightarrow \mathcal{P} \models \varphi.$$

The above theorem leads to a model checking algorithms that iteratively refines the approximations of $Reach(l)$ and $Recur(l)$ for a fair location l until one of the termination conditions becomes valid.

3 Avoid Loop Unrolling: Implementation

As the driving force behind such iterative refinement we propose using PDR, lifted according to the following points:

- refine $Recur(l)$ sets using *counter-examples to recurrence* (CtR) and $Reach(l)$ sets using counter-examples to induction;
- localise the refinements to program locations;
- extend the method to the bit-vector theory (similarly as in [4]).

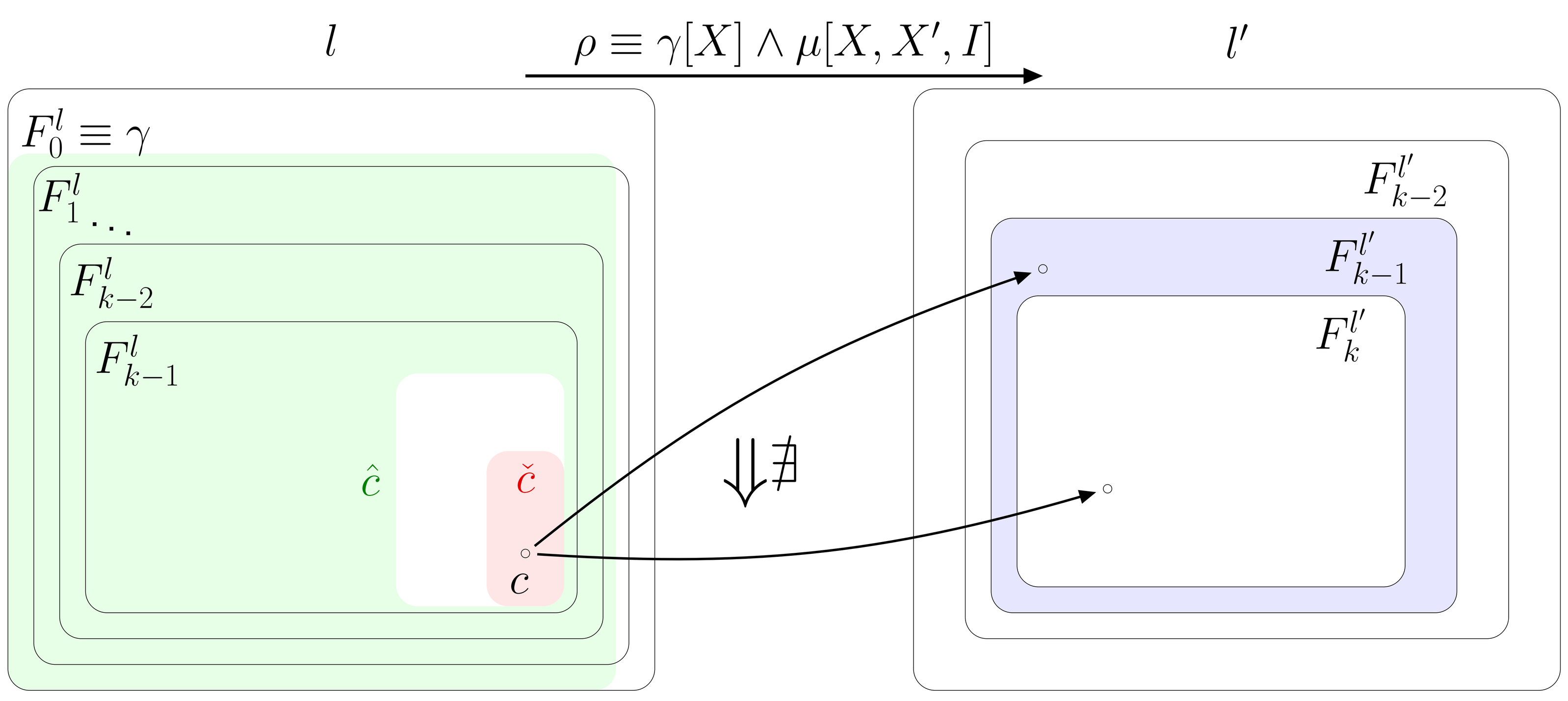
Under-approximation Search for under-approximations (w.r.t. a fair location l_F) has two cooperating stages:

1. standard PDR with $P \equiv \neg \mathtt{at}\_l_F$ describing the safe states $\Longrightarrow x \in Reach(l_F)$
2. PDR with $P \equiv \neg(\mathtt{at}\_l_F \wedge x)$, initial state set to (l_F, x), and limited to the strongly connected component containing l_F.

Over-approximation The standard definition of the recurrent set requires quantifier alternation: Let X, X' be the sets of program variables for the current and the next state and I the input variables. Then G is a recurrent set w.r.t. to the cycle relation R iff

$$\forall X \exists X' \exists I,\ G[X] \Rightarrow R[X, X', I] \wedge G'[X'].$$

It follows that a CtR is any evaluation without either a predecessor or a successor. The figure below illustrates the search for single-step CtRs.



Description The task is to refine F^l based on the new $F^{l'}$, assuming the invariant

$$\forall i < k, \forall X \exists X' \exists I,\ F_i^l \Rightarrow \rho \wedge F_i^{l'}$$

- initialise $\quad F_k^l := F_{k-1}^l$
- find potential CtR $\quad \mathcal{M} := \mathtt{sat}(F_k^l \wedge \rho \wedge (F_{k-1}^{l'} \wedge \neg F_k^{l'})) \quad c, c' := \mathcal{M}_{|X}, \mathcal{M}_{|X'}$
- check c $\quad q := \mathtt{sat}(c \wedge \rho \wedge F_k^{l'})$
  - $q = sat$ $\begin{cases} \rho[X, X', I] & \hat{c} \wedge F_{k-1}^l \wedge \rho \wedge c' \ldots sat \\ \rho[X, X'] & \hat{c} \wedge F_{k-1}^l \wedge \rho \wedge F_k^{l'} \ldots sat \end{cases} \quad \hat{c} \Rightarrow \neg c$
  - $q = unsat$ $\quad \check{c} \wedge \rho \wedge F_k^{l'} \ldots unsat \quad c \Rightarrow \check{c}$