Simple is Better: Efficient Bounded Model Checking for Past LTL

Timo Latvala[1], Armin Biere[2], Keijo Heljanko[1], and Tommi Junttila[1]

`Timo.Latvala@hut.fi`

[1]Laboratory for Theoretical Computer Science

Helsinki University of Technology

[2]Institute for Formal Models and Verification

Johannes Kepler University

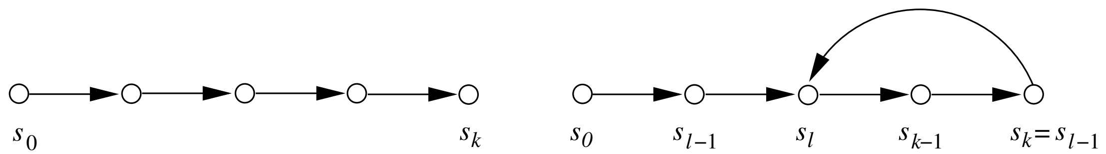
Introduction

- Bounded model checking (BMC) is an efficient way of implementing *symbolic model checking*.
- Alleviate state explosion by representing the state space implicitly.
- BMC: given a system model M, a temporal logic specification ψ, and bound k create a Boolean formula which is satisfiable *iff* M has a witness, of length k, to $\neg\psi$.
- Basic form: $|[M]|_k \wedge |[\neg\psi]|_k$

Introduction

- Bounded model checking (BMC) is an efficient way of implementing *symbolic model checking*.
- Alleviate state explosion by representing the state space implicitly.
- BMC: given a system model M, a temporal logic specification ψ, and bound k create a Boolean formula which is satisfiable *iff* M has a witness, of length k, to $\neg\psi$.
- Basic form: $|[M]|_k \wedge |[\neg\psi]|_k$

BMC Basics



(a) no loop (b) (k,l)–loop

- Consider (symbolically) all (k,l)-loops of the system.
  - write constraints $_l|[\psi]|_k$ such that the formula ψ is satisfiable *iff* the selected (k,l)-loop is a valid counterexample.
- All possible loops:

$$\bigvee_{l=1}^{k} {}_l|[\psi]|_k$$

BMC: Pros and Cons

+ Boolean formulas can be more compact BDDs.
+ Leverages efficient SAT-solver technology.
+ Short counterexamples.
- Basic method is incomplete.
- Not always better than BDD-based methods.

PLTL

- A logic with the usual temporal operators Until, Release, Next and past operators Since, Historically, Once. . . .
- Exponentially more succinct than LTL.
- Considered more intuitive than LTL: "Acknowledgement are issued only upon requests".
- $\mathbf{G}\,(ack \Rightarrow \mathbf{Y}\,(\neg ack\,\mathbf{S}\,req))$ vs $(req\mathbf{R}\neg ack) \wedge \mathbf{G}\,(ack \Rightarrow (ack \vee \mathbf{X}\,(req\mathbf{R}\neg ack)))$.
- Uses: requirement engineering, specification, runtime verification.

PLTL Properties

x	0	1	2	3	4	5	2	3	4	5	2	3	4	5	2	3	4	5
time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

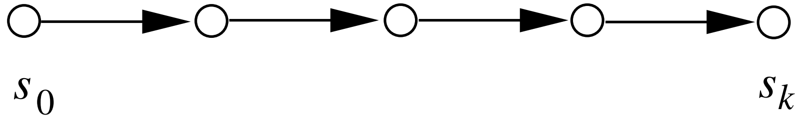
- Simple counter with an execution $\pi = (012)(3452)^{\omega}$.
- $\mathbf{F}\psi : \psi = (x = 3 \wedge \mathbf{O}\,(x = 4 \wedge \mathbf{O}\,(x = 5)))$.
- $\pi^{8} \models (x = 4) \wedge \mathbf{O}\,(x = 5)$. $\pi^{8} \not\models \psi$.
- $\pi^{11} \models \psi$.
- **Prop. 1:** PLTL can distinguish between unrollings of the loop *only* up to the past depth $\delta(\psi)$ of the formula.

A New Encoding

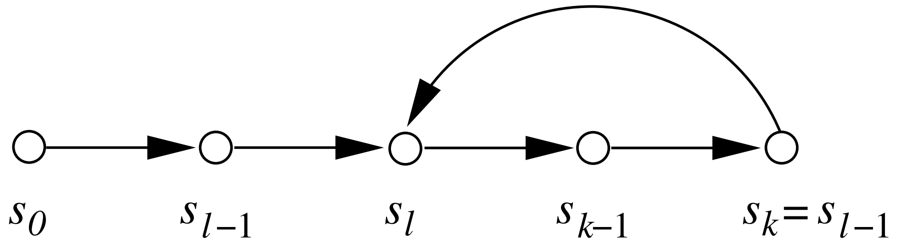
- Structure: $|[M]|_k \wedge |[LoopConstraints]|_k \wedge |[\neg\psi]|_k$.
- $|[M]|_k$: paths of length k.
- $|[LoopConstraints]|_k$: select a (k,l)-loop.
- $|[\neg\psi]|_k$: check that the selected (k,l)-loop is a witness to $\neg\psi$.

Selecting (k,l)-loops

- Encoding should select non-deterministically select a k-length lasso-shaped path.
- Introduce k fresh *loop selector variables* l_i:
  - $l_i \Rightarrow (s_{l-1} = s_k)$.
- Allow *at most one* loop selector to be true



(a) no loop

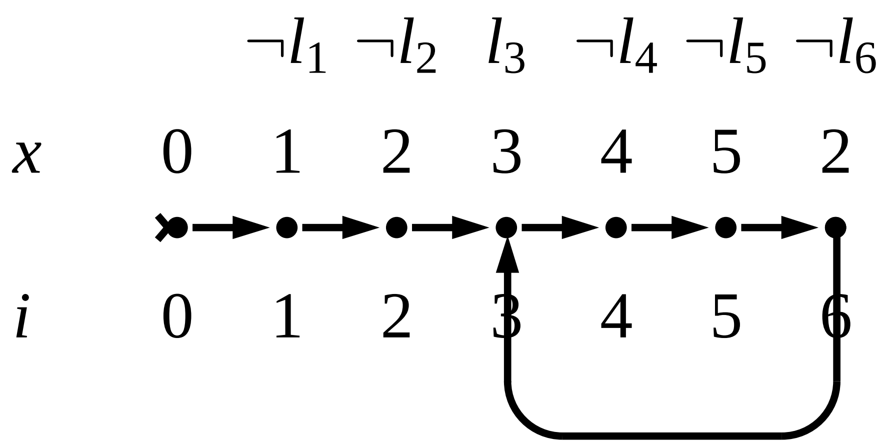


(b) (k,l)–loop

Selecting (k,l)-loops II

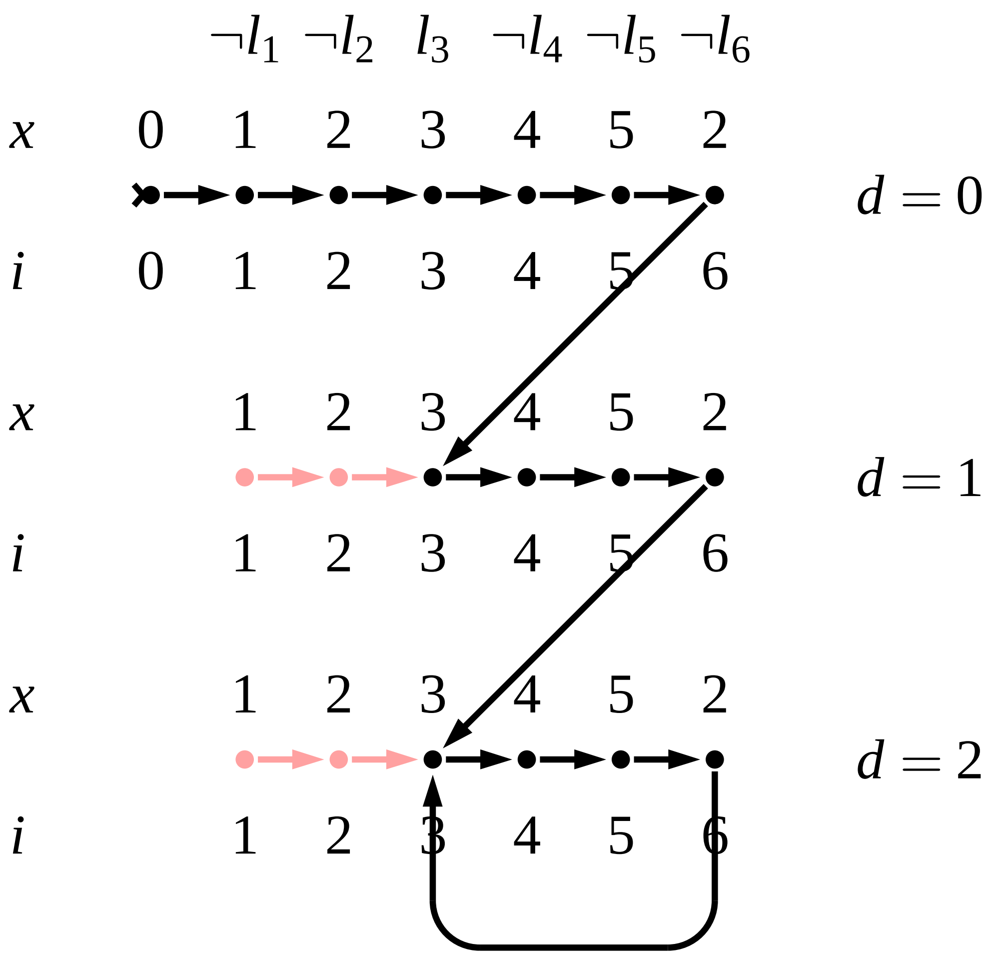
$$
\begin{aligned}
|[M]|_k &= I(s_0) \wedge \bigwedge_{i=1}^{k} T(s_{i-1}, s_i) \\
|[LoopConstraints]|_k &\Leftrightarrow Loop_k \wedge AtMostOne_k \\
Loop_k &\Leftrightarrow \bigwedge_{i=1}^{k} (l_i \Rightarrow (s_{i-1} = s_k)) \\
AtMostOne_k &\Leftrightarrow \bigwedge_{i=1}^{k} (SmallerExists_i \Rightarrow \neg l_i) \\
SmallerExists_1 &\Leftrightarrow \bot \\
SmallerExists_{i+1} &\Leftrightarrow SmallerExists_i \vee l_i, \text{ where } 0 < i \leq k
\end{aligned}
$$

Virtual Unrolling I



- $\pi = 012(3452)^\omega$
- Past formulas inside the loop see different pasts depending on how many times we go back inside the loop.
- $\pi \models \mathbf{GFZZZ}\,(x = 0)$?
- Solution: unroll the loop.

Virtual Unrolling II



- Virtual unrolling for $\delta(\psi) = 2$.
- Let each past subformula see a sufficiently unrolled (k, l)-loop.
- Virtually unroll: going to higher k:s is expensive.

Encoding Basics

$|[\psi]|_i^d$: two parameters, unrolling depth d and position i.

$:=$	$0 \leq i \leq k$
$\vert[p]\vert_i^d$	p_i
$\vert[\neg p]\vert_i^d$	$\neg p_i$
$\vert[\psi_1 \vee \psi_2]\vert_i^d$	$\vert[\psi_1]\vert_i^d \vee \vert[\psi_2]\vert_i^d$
$\vert[\psi_1 \wedge \psi_2]\vert_i^d$	$\vert[\psi_1]\vert_i^d \wedge \vert[\psi_2]\vert_i^d$

Encoding Until

- The encoding of Until and Release are based on their fixpoint characterisations.
- Use an auxiliary encoding $\langle\langle\cdot\rangle\rangle$ to compute an approximation of the fixpoint.
- The approximate values are refined to exact by the $|[\cdot]|$-encoding.
- Virtual unrolling by copying the LTL subformulas of the path $\delta(\psi)$ times.

Encoding Until II

φ	$0 \le d < \delta(\varphi), 0 \le i < k$	$0 \le d < \delta(\varphi), i = k$
$\|[\psi_1 \mathbf{U} \psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \vee \left(\|[\psi_1]\|_i^d \wedge \|[\psi_1 \mathbf{U} \psi_2]\|_{i+1}^d \right)$	$\bigvee_{j=1}^{k} \left(l_j \wedge \|[\psi_1 \mathbf{U} \psi_2]\|_j^{d+1} \right)$

Encoding Until III

φ	$d = \delta(\varphi), 0 \leq i < k$	$d = \delta(\varphi), i = k$
$\lvert[\psi_1 \mathbf{U} \psi_2]\rvert_i$	$\lvert[\psi_2]\rvert_i^d \vee \left(\lvert[\psi_1]\rvert_i^d \wedge \lvert[\psi_1 \mathbf{U} \psi_2]\rvert_{i+1}^d\right)$	$\bigvee_{j=1}^{k} \left(l_j \wedge \langle\langle \psi_1 \mathbf{U} \psi_2 \rangle\rangle_j^d\right)$
$\langle\langle \psi_1 \mathbf{U} \psi_2 \rangle\rangle_i^d$	$\lvert[\psi_2]\rvert_i \vee \left(\lvert[\psi_1]\rvert_i \wedge \langle\langle \psi_1 \mathbf{U} \psi_2 \rangle\rangle_{i+1}^d\right)$	$\lvert[\psi_2]\rvert_i^d$

Encoding Since

- Use of virtual unrolling makes encoding past operators fairly straightforward based on their fixpoint characterisations.
- Loop selector variables makes choice between going back inside the loop or back to the origin easy to express.

Encoding Since II

φ	$d = 0, i = 0$	$d = 0, 1 \le i \le k$
$\lvert[\psi_1 \mathbf{S}\, \psi_2]\rvert_i^d$	$\lvert[\psi_2]\rvert_i^d$	$\lvert[\psi_2]\rvert_i^d \vee \left(\lvert[\psi_1]\rvert_i^d \wedge \lvert[\psi_1 \mathbf{S}\, \psi_2]\rvert_{i-1}^d\right)$

φ	$1 \le d \le \delta(\varphi), 1 \le i \le k$
$\lvert[\psi_1 \mathbf{S}\, \psi_2]\rvert_i^d$	$\lvert[\psi_2]\rvert_i^d \vee \left(\lvert[\psi_1]\rvert_i^d \wedge \left(\left(l_i \wedge \lvert[\varphi]\rvert_k^{d-1}\right) \vee \left(\neg l_i \wedge \lvert[\varphi]\rvert_{i-1}^d\right)\right)\right)$

Main Result

Theorem
The size of $|[M, \psi, k]|$ seen as Boolean circuit is of the order $O(|I| + k \cdot |T| + k \cdot |\psi| \cdot \delta(\psi))$.

Since $\delta(\psi)$ can be in $O(|\psi|)$ the encoding has a worst case quadratic complexity w.r.t. $|\psi|$. When $\delta(\psi)$ is fixed (e.g. LTL), the encoding is linear in all parameters.

Properties of the Encoding

- Future fragment collapses to our LTL encoding (FMCAD 2004).
- Unique model property.
- Monotonic circuit.
- Simple and easy to understand.
- Detects *minimal length* counterexamples.

Related Work

- Original BMC encoding: Biere et al. (TACAS 1999).
- First BMC encoding with past: Benedetti and Cimatti (TACAS 2003)
- Fixpoint encoding with past: Cimatti et al. (FMCAD 2004).

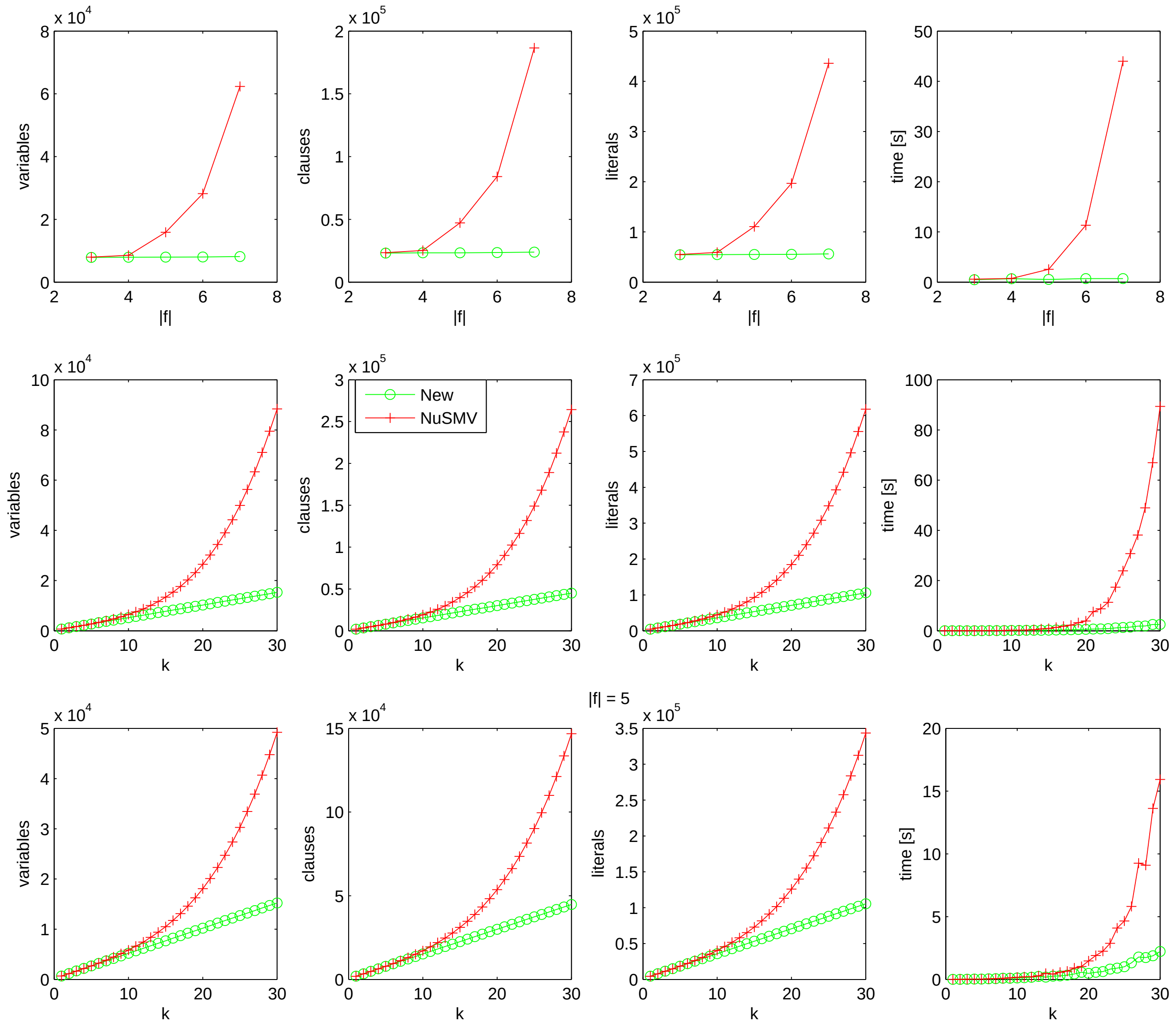
Experiments

- Random formulae on small random Kripke structures.
- Formula sizes between 3-7, and k from $0-30$.
- A few real-life examples.
- Compare with the encoding of NuSMV (Benedetti and Cimatti).
- Measure: number of variables, clauses and literals in the CNF encoding, time to solve instance.

Benchmarks

M	k	NuSMV				New			
		vars	*clauses*	*time*	Σ*time*	*vars*	*clauses*	*time*	Σ*time*
abp	16	25,175	74,208	104	342	22,827	67,116	52.5	269
brp	10	14,115	41,228	0.9	2.5	8,961	25,736	0.7	2.2
	15	30,225	89,218	4.6	15.9	13,346	38,536	1.5	7.5
	20	56,935	169,008	19.2	75.6	17,731	51,336	3.2	19.7
dme	10	49,776	139,740	10.3	15.1	28,855	76,947	6.3	17.5
	15	139,071	404,485	98.9	171	42,685	115,282	15.5	70.2
	20	346,166	1,022,630	1,017	1,812	56,515	153,617	41.2	214
pci	10	81,285	242,133	96.7	188	60,456	179,616	69.8	151
	15	159,885	477,358	2,441	5,408	90,611	269,491	888	2,422
	18	227,357	679,429	2,557	19,119	108,704	323,416	867	11,992
srg5	10	137,710	412,952	53.6	90.7	1,655	4,757	0.0	0.1
	18	1,264,988	3,794,698	14,914	33,708	2,999	8,677	0.2	0.9
	30	N/A	N/A	N/A	N/A	5,015	14,557	0.7	6.6

Benchmarks II



Conclusions and Future Work

- An efficient BMC encoding for PLTL.
- Implementation available from `http://www.tcs.hut.fi/~timo/vmcai2005/`

Future work:

- Use monotonicity
- Incremental BMC