Advanced Tutorial on Bounded Model Checking (BMC)

ACSD'06 - ATPN'06

26th of June 2006

Keijo Heljanko and Tommi Junttila

{Keijo.Heljanko,Tommi.Junttila}@tkk.fi

Part II: Encoding Temporal Logics in BMC

Outline

- Kripke Structures and LTL
- Simple BMC for LTL
- BMC and Incremental SAT
- Making BMC Complete
- Simple BMC for LTL with Past Operators

Temporal Logics

- Beyond reachability properties
- We will consider linear time temporal logics LTL and LTL with past operators (PLTL)
- No quantification over paths (branching) like in e.g. CTL

Kripke Structures

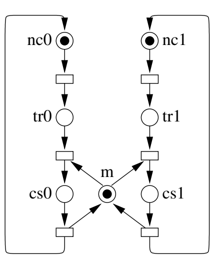
- Kripke structures are a fully modelling language independent way of representing the behaviour of parallel and distributed systems.
- Kripke structures are graphs which describe all the possible executions of the system, where all internal state information has been hidden, except for some interesting atomic propositions.

Formal Definition

- Let AP be a finite set of atomic propositions. A Kripke structure is a four-tuple $M = (S, s_{init}, T, L)$, where
  - S is a finite set of states,
  - $s_{init} \in S$ is the initial state (marked with a wedge),
  - $T \subseteq S \times S$ is a total transition relation, ($(s, s') \in T$ is drawn as an arc from s to s'), and
  - $L : S \to 2^{AP}$ is a valuation, i.e. a function which maps each state to those atomic propositions which hold in that state.

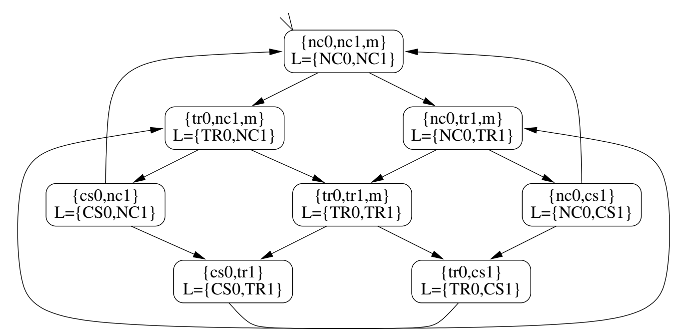
Running Example

- A simple 1-safe Petri net for mutex



Running Example: Kripke Structure

- $AP = \{\mathrm{NC0}, \mathrm{TR0}, \mathrm{CS0}, \mathrm{NC1}, \mathrm{TR1}, \mathrm{CS1}\}$



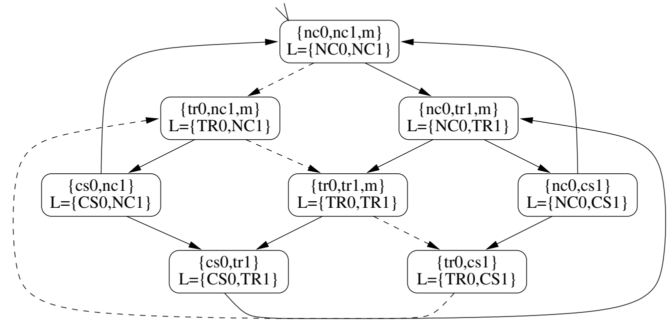
Paths and (k, l)-Loops

- A path in a Kripke structure $M = (S, s_{init}, T, L)$ is an infinite sequence $\pi = s_0 s_1 \ldots$ of states in S such that
  - $s_0 = s_{init}$, and
  - $T(s_i, s_{i+1})$ holds for all $i \geq 0$
- A path $\pi = s_0 s_1 \ldots$ is a (k, l)-loop if $\pi = (s_0 s_1 \ldots s_{l-1})(s_l \ldots s_k)^\omega$ such that $0 < l \leq k$ and $s_{l-1} = s_k$
- If π is a (k, l)-loop, then it is a $(k+1, l+1)$-loop

Running Example: Paths

- The dashed path in the figure is a $(4, 2)$-loop as it equals to

$$\{\mathrm{nc0}, \mathrm{nc1}, \mathrm{m}\}\ \{\mathrm{tr0}, \mathrm{nc1}, \mathrm{m}\}\ (\{\mathrm{tr0}, \mathrm{tr1}, \mathrm{m}\}\ \{\mathrm{tr0}, \mathrm{cs1}\}\ \{\mathrm{tr0}, \mathrm{nc1}, \mathrm{m}\})^\omega$$



LTL Syntax

- Each $p \in AP$ is an LTL formula
- If ψ_1 and ψ_2 are LTL formulae, then the following are LTL formulae:

$\neg\psi_1$	negation
$\psi_1 \vee \psi_2$	disjunction
$\psi_1 \wedge \psi_2$	conjunction
$\mathbf{X}\,\psi_1$	"next"
$\mathbf{F}\,\psi_1$	"finally" (or "eventually")
$\mathbf{G}\,\psi_1$	"globally" (or "always")
$\psi_1\,\mathbf{U}\,\psi_2$	"until"
$\psi_1\,\mathbf{R}\,\psi_2$	"release"

Examples of LTL formulae

- Invariance:
  $\mathbf{G}\,\neg(\mathrm{CS0} \wedge \mathrm{CS1})$
- Reachability:
  $\mathbf{F}\,(\mathrm{CS0})$
- Process 0 always finally leaves the critical section:
  $\mathbf{G}\,(\mathrm{CS0} \Rightarrow \mathbf{F}\,(\neg\mathrm{CS0}))$
- "Justice" fairness (infinitely often):
  $\mathbf{G}\,\mathbf{F}\,(\mathrm{CS0})$
- "Weak" fairness:
  $(\mathbf{F}\,\mathbf{G}\,(\mathrm{TR0})) \Rightarrow (\mathbf{G}\,\mathbf{F}\,(\mathrm{CS0}))$
- "Strong" fairness:
  $(\mathbf{G}\,\mathbf{F}\,(\mathrm{TR0})) \Rightarrow (\mathbf{G}\,\mathbf{F}\,(\mathrm{CS0}))$

LTL Syntax

- In principle, it is sufficient to consider only e.g.

$$\neg, \vee, \mathbf{X}, \mathbf{U}$$

as all the other operators can be derived from them

- For convenience, and to define positive normal form, we use the other operators, too
- The following abbreviations are common:

$$\begin{aligned}
\mathbf{1} &\equiv p \vee (\neg p) \\
\mathbf{0} &\equiv \neg \mathbf{1} \\
\psi_1 \Rightarrow \psi_2 &\equiv (\neg \psi_1) \vee \psi_2 \\
\psi_1 \Leftrightarrow \psi_2 &\equiv (\psi_1 \Rightarrow \psi_2) \wedge (\psi_2 \Rightarrow \psi_1)
\end{aligned}$$

LTL Syntax

- The following equations hold:

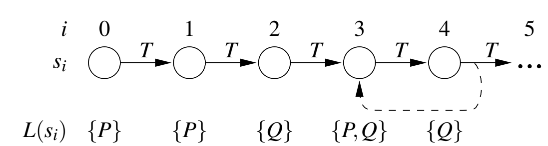
$$\begin{aligned}
\mathbf{F}\,\psi_1 &\Leftrightarrow \mathbf{1}\,\mathbf{U}\,\psi_1 \\
\mathbf{G}\,\psi_1 &\Leftrightarrow \mathbf{0}\,\mathbf{R}\,\psi_1 \\
\psi_1 \wedge \psi_2 &\Leftrightarrow \neg((\neg \psi_1) \vee (\neg \psi_2)) \\
\psi_1 \vee \psi_2 &\Leftrightarrow \neg((\neg \psi_1) \wedge (\neg \psi_2)) \\
\mathbf{G}\,\psi_1 &\Leftrightarrow \neg \mathbf{F}\,(\neg \psi_1) \\
\mathbf{F}\,\psi_1 &\Leftrightarrow \neg \mathbf{G}\,(\neg \psi_1) \\
\psi_1\,\mathbf{U}\,\psi_2 &\Leftrightarrow \neg(\neg \psi_1\,\mathbf{R}\,\neg \psi_2) \\
\psi_1\,\mathbf{R}\,\psi_2 &\Leftrightarrow \neg(\neg \psi_1\,\mathbf{U}\,\neg \psi_2)
\end{aligned}$$

Semantics of LTL

- Let $\pi = s_0 s_1 \ldots$ be a path with labelling $L(s_i) \in 2^{AP}$
- The relation $\pi^i \models \psi$ for "ψ holds at time point i in π":

$$\begin{aligned}
\pi^i \models \psi &\Leftrightarrow \psi \in L(s_i) \text{ for } \psi \in AP \\
\pi^i \models \neg\psi &\Leftrightarrow \pi^i \not\models \psi \\
\pi^i \models \psi_1 \vee \psi_2 &\Leftrightarrow \pi^i \models \psi_1 \text{ or } \pi^i \models \psi_2 \\
\pi^i \models \psi_1 \wedge \psi_2 &\Leftrightarrow \pi^i \models \psi_1 \text{ and } \pi^i \models \psi_2 \\
\pi^i \models \mathbf{X}\,\psi &\Leftrightarrow \pi^{i+1} \models \psi \\
\pi^i \models \mathbf{F}\,\psi_1 &\Leftrightarrow \exists n \geq i : \pi^n \models \psi_1 \\
\pi^i \models \mathbf{G}\,\psi_1 &\Leftrightarrow \forall n \geq i : \pi^n \models \psi_1 \\
\pi^i \models \psi_1\,\mathbf{U}\,\psi_2 &\Leftrightarrow \exists n \geq i : (\pi^n \models \psi_2 \wedge \forall i \leq j < n : \pi^j \models \psi_1) \\
\pi^i \models \psi_1\,\mathbf{R}\,\psi_2 &\Leftrightarrow (\forall n \geq i : \pi^n \models \psi_2) \vee \\
& (\exists n \geq i : \pi^n \models \psi_1 \wedge \forall i \leq j \leq n : \pi^j \models \psi_2)
\end{aligned}$$

Semantics of LTL



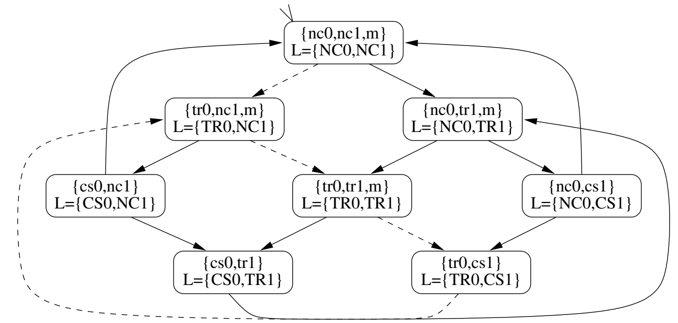
- $\pi^0 \models P$, $\pi^0 \not\models Q$, $\pi^2 \models Q$
- $\pi^0 \models P\,\mathbf{U}\,Q$, $\pi^0 \not\models Q\,\mathbf{R}\,P$
- $\pi^0 \models \mathbf{F}\,Q$, $\pi^0 \not\models \mathbf{G}\,P$
- $\pi^2 \models \mathbf{G}\,Q$
- $\pi^0 \models \mathbf{F}\,\mathbf{G}\,Q$
- $\pi^0 \models \mathbf{G}\,\mathbf{F}\,P$

Semantics of LTL

- We write $\pi \models \psi$ if $\pi^0 \models \psi$ and say that π is a witness path for ψ
- An LTL formula ψ holds in a Kripke structure $M = (S, s_{init}, T, L)$ if $\pi \models \psi$ for each path π in M
- Model checking problem: find whether $M \models \psi$
- Dually: is there a counter-example path π in M such that $\pi \models \neg\psi$?
  - If there is, then $M \not\models \psi$.
  - Otherwise, $M \models \psi$.

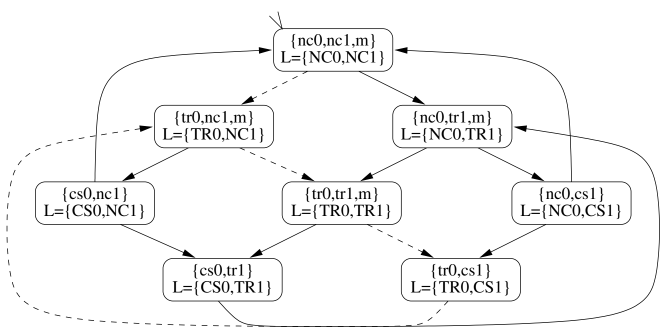
Running Example: LTL

- The dashed path below is a witness for $\mathbf{G}\,(\neg \mathrm{CS0})$ and thus a counter-example for $\neg\mathbf{G}\,(\neg \mathrm{CS0}) \equiv \mathbf{F}\,(\mathrm{CS0})$



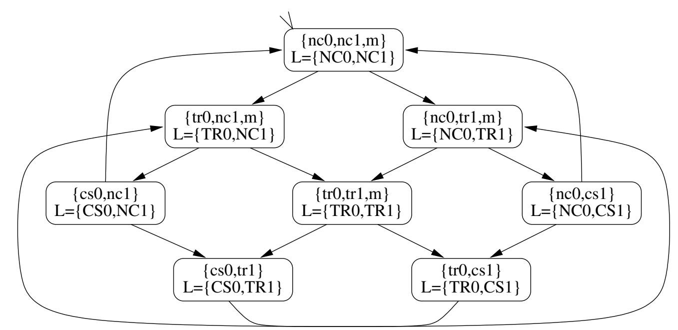
Running Example: LTL

- The dashed path below is a witness for $\mathbf{F}\,(\mathbf{G}\,(\mathrm{TR0}))$ and thus a counter-example for $\mathbf{G}\,(\mathbf{F}\,(\neg\mathrm{TR0}))$



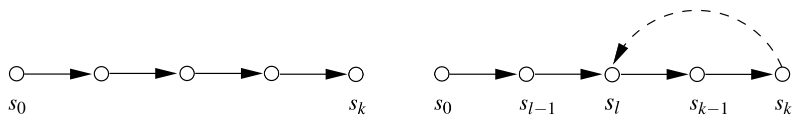
Running Example: LTL

- There is no witness for $\mathbf{F}\,(\mathrm{CS0} \wedge \mathrm{CS1})$ and thus $\neg\mathbf{F}\,(\mathrm{CS0} \wedge \mathrm{CS1}) \equiv \mathbf{G}\,\neg(\mathrm{CS0} \wedge \mathrm{CS1})$ holds



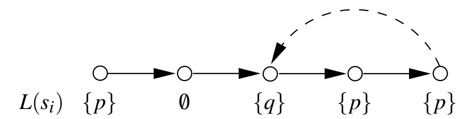
Bounded Paths

- BMC considers k-paths, i.e., bounded paths with k transitions
- A k-path can represent
  - all its infinite extensions (the "no loop" case), or
  - a (k,l)-loop $s_0 \ldots s_{l-1}(s_l \ldots s_k)^\omega$ if $s_k = s_{l-1}$ for some $1 \leq l \leq k$



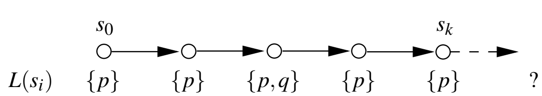
(a) no loop (b) (k,l)-loop

Bounded Paths and LTL



- Consider the $(4,2)$-loop π above
- We can check whether $\pi \models \psi$ for any ψ
- For example, $\pi \models \mathbf{F}\,(p\,\mathbf{U}\,q)$

Bounded Paths and LTL



- Consider the no-loop case above
- We know that $\pi \models \mathbf{F}\,q$ for each infinite extension π
- But we don't know whether $\pi \models \mathbf{G}\,p$ for any infinite extension π
- To formalize this, we need *bounded* semantics of LTL

Positive Normal Form for LTL

- From now on, we assume that negations can only appear in front of atomic propositions
- Every LTL formula can be translated to equivalent positive normal form formula by using:

$$
\begin{aligned}
\neg(\psi_1 \vee \psi_2) &\equiv (\neg\psi_1) \wedge (\neg\psi_2) \\
\neg(\psi_1 \wedge \psi_2) &\equiv (\neg\psi_1) \vee (\neg\psi_2) \\
\neg(\neg\psi) &\equiv \psi \\
\neg(\mathbf{X}\,\psi) &\equiv \mathbf{X}\,(\neg\psi) \\
\neg(\mathbf{F}\,\psi) &\equiv \mathbf{G}\,(\neg\psi) \\
\neg(\mathbf{G}\,\psi) &\equiv \mathbf{F}\,(\neg\psi) \\
\neg(\psi_1\,\mathbf{U}\,\psi_2) &\equiv (\neg\psi_1)\,\mathbf{R}\,(\neg\psi_2) \\
\neg(\psi_1\,\mathbf{R}\,\psi_2) &\equiv (\neg\psi_1)\,\mathbf{U}\,(\neg\psi_2)
\end{aligned}
$$

Bounded Semantics of LTL

- Given a path $\pi = s_0 s_1 \ldots$ and a bound $k \geq 0$, $\pi \models_k \psi$ iff (i) π is a (k,l)-loop and $\pi^0 \models \psi$, or (ii) $\pi^0 \models_{nl} \psi$, where

$$
\begin{array}{lcl}
\pi^i \models_{nl} p & \Leftrightarrow & p \in L(s_i) \text{ for } p \in AP \\
\pi^i \models_{nl} \neg p & \Leftrightarrow & p \notin L(s_i) \text{ for } p \in AP \\
\pi^i \models_{nl} \psi_1 \vee \psi_2 & \Leftrightarrow & \pi^i \models_{nl} \psi_1 \text{ or } \pi^i \models_{nl} \psi_2 \\
\pi^i \models_{nl} \psi_1 \wedge \psi_2 & \Leftrightarrow & \pi^i \models_{nl} \psi_2 \text{ and } \pi^i \models_{nl} \psi_2 \\
\pi^i \models_{nl} \mathbf{X}\,\psi_1 & \Leftrightarrow & i < k \text{ and } \pi^{i+1} \models_{nl} \psi_1 \\
\pi^i \models_{nl} \mathbf{F}\,\psi_1 & \Leftrightarrow & \exists i \leq n \leq k : \pi^n \models_{nl} \psi_1 \\
\pi^i \models_{nl} \mathbf{G}\,\psi_1 & \Leftrightarrow & \mathbf{0} \\
\pi^i \models_{nl} \psi_1 \,\mathbf{U}\, \psi_2 & \Leftrightarrow & \exists i \leq n \leq k : (\pi^n \models_{nl} \psi_2 \wedge \forall i \leq j < n : \pi^j \models_{nl} \psi_1) \\
\pi^i \models_{nl} \psi_1 \,\mathbf{R}\, \psi_2 & \Leftrightarrow & \exists i \leq n \leq k : (\pi^n \models_{nl} \psi_1 \wedge \forall i \leq j \leq n : \pi^j \models_{nl} \psi_2)
\end{array}
$$

Bounded Semantics of LTL

- $\models_k$ under-approximates $\models$.
- If $\pi \models_k \psi$, then $\pi \models \psi$.
- For each ultimately periodic path π there is a k such that π is a (k,l)-loop and thus $\pi \models \psi$ iff $\pi \models_k \psi$.
- If $\pi \models_k \psi$, then $\pi \models_{k+1} \psi$.

BMC for LTL

BMC Encoding for LTL

- Given a symbolic representation of a Kripke structure M, a LTL formula ψ, and a bound k
- Goal: build a formula $|[M,\psi,k]|$ that is satisfiable iff M has a path π such that $\pi \models_k \psi$

BMC Encoding for LTL

- The generic form of $|[M, \psi, k]|$ is

$$|[M]|_k \wedge |[\psi, k]|_0$$

- As before, $|[M]|_k \equiv I(s_0) \wedge \bigwedge_{i=1}^{k} T(s_{i-1}, s_i)$ encodes paths by unrolling transition relation k times
- $|[\psi, k]|_0$ constraints paths to be witnesses for ψ under the bounded semantics

BMC for LTL: Our Approach

- Latvala, T., Biere, A., Heljanko, K., and Junttila, T.: *Simple Bounded LTL Model Checking*. FMCAD'04.
  - Non-incremental, compact BMC encoding for LTL with no past operators
- Latvala, T., Biere, A., Heljanko, K., and Junttila, T.: *Simple Is Better: Efficient Bounded Model Checking for Past LTL*. VMCAI'05.
  - Non-incremental, compact BMC encoding for LTL with past operators

BMC for LTL: Our Approach

- Heljanko, K., Junttila, T., and Latvala, T.: *Incremental and Complete Bounded Model Checking for Full PLTL*. CAV'05.
- Latvala, T.: *Automata-theoretic and bounded model checking for linear temporal logic*. Doctoral dissertation, Helsinki University of Technology, 2005.

BMC for LTL: Some Related Work

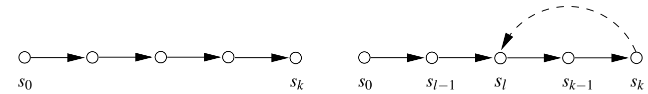
- Biere, A., Cimatti, A., Clarke, E., and Zhu, Y.: Symbolic Model Checking without BDDs. TACAS'99.
  - First LTL to SAT encoding
- Cimatti, A., Pistore, M., Roveri, M., and Sebastiani, R.: Inproving the encoding of LTL model checking into SAT. VMCAI'02.
  - Improvements to the above encoding

BMC for LTL: Some Related Work

- Benedetti, M. and Cimatti, A.: Bounded Model Checking for Past LTL. TACAS’03.
  - Encoding for Past LTL
- Schuppan, V., and Biere, A.: Shortest counterexamples for symbolic model checking of LTL. TACAS’05
  - Our VMCAI translation + liveness-to-safety + BDDs

Original BMC encoding

- Basic encoding form: $|[M]|_k \wedge |[\psi, k]|$



(a) no loop (b) (k,l)-loop

- Basic idea: $|[\psi, k]| \equiv {}_{-}|[\psi, k]|_0 \vee \bigvee_{l=1}^{k} {}_{l}|[\psi, k]|_0$, where
  - ${}_{-}|[\psi, k]|_0$ evaluates ψ in the no loop case
  - ${}_{l}|[\psi, k]|_0$ evaluates ψ in the (k,l)-loop case
- Size: $\Omega(|I| + k \cdot |T| + k^2 \cdot |\psi|)$

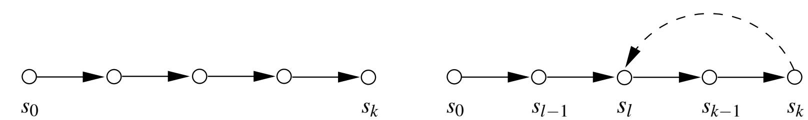
Simple BMC Encoding for LTL

- Goal: build a formula $|[M, \psi, k]|$ that is satisfiable iff M has a path π such that $\pi \models_k \psi$
- The generic form of our translation is

$$|[M]|_k \wedge |[\text{LoopConstraints}]|_k \wedge |[\text{LastStateConstraints}]|_k \wedge |[\psi, k]|_0$$

- As before, $|[M]|_k \equiv I(s_0) \wedge \bigwedge_{i=1}^{k} T(s_{i-1}, s_i)$
- Seen as a Boolean circuit, $|[M, \psi, k]|$ is of size $O(|I| + k \cdot |T| + k \cdot |\psi|)$

Loop Constraints



(a) no loop (b) (k,l)-loop

- Non-deterministically select a (k,l)-loop or the no loop case
- Introduce *free loop selector variables* l_i:
  - Constrain $l_i \Rightarrow (s_{i-1} = s_k)$
- Allow *at most one* loop selector to be true

Loop Constraints

	$\lvert[\text{LoopConstraints}]\rvert_k$		
Base	l_0	$\Leftrightarrow$	$\mathbf{0}$
	InLoop_0	$\Leftrightarrow$	$\mathbf{0}$
$1 \le i \le k$	l_i	$\Rightarrow$	$(s_{i-1} = s_k)$
	InLoop_i	$\Leftrightarrow$	$\text{InLoop}_{i-1} \lor l_i$
	l_i	$\Rightarrow$	$\neg\text{InLoop}_{i-1}$
	LoopExists	$\Leftrightarrow$	InLoop_k

- InLoop_i is true iff the i:th state belongs to the selected loop
- At most one l_i is allowed to be true
- LoopExists is true iff a (k,i)-loop was selected

Illustration of the Encoding

- Mutex example, $k = 3$, no loop
- Finite path prefix
  $\{\text{nc0}, \text{nc1}, \text{m}\}\ \{\text{tr0}, \text{nc1}, \text{m}\}\ \{\text{tr0}, \text{tr1}, \text{m}\}\ \{\text{tr0}, \text{cs1}\}$

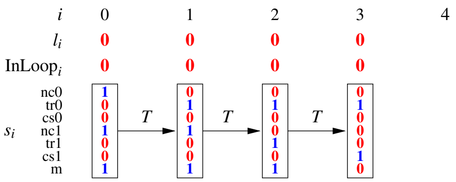
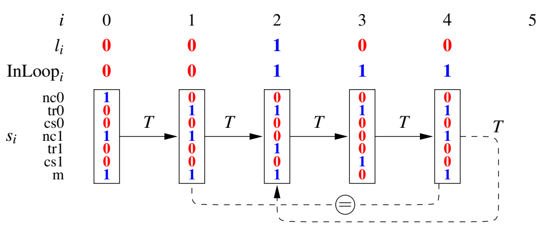


Illustration of the Encoding

- Mutex example, $k = 4$, $l_2 = \mathbf{1}$
- The $(4,2)$-loop
  $\{\text{nc0}, \text{nc1}, \text{m}\}\ \{\text{tr0}, \text{nc1}, \text{m}\}\ (\{\text{tr0}, \text{tr1}, \text{m}\}\ \{\text{tr0}, \text{cs1}\}\ \{\text{tr0}, \text{nc1}, \text{m}\})^\omega$



Encoding LTL: Subformula Variables

- For each subformula φ of ψ, introduce a variable $\lvert[\varphi]\rvert_i$ where $i \in \{0, 1, \ldots, k, k+1\}$
- $\lvert[\varphi]\rvert_i$ evaluates the value of the subformula φ at time step i
- Thus $\lvert[\psi]\rvert_0$ evaluates whether $\pi \models_k \psi$ under the selected (k,l)-loop/no loop case
- The $k+1$th index is the "future" index, the successor of the kth index

Encoding LTL: Last State Constraints

- The no-loop case: force "pessimistic" future
- The (k,i)-loop case: connect the future state $k+1$ to the loop state i

	$\lvert[\text{LastStateConstraints}]\rvert_k$
Base	$\neg\text{LoopExists} \Rightarrow (\lvert[\phi]\rvert_{k+1} \Leftrightarrow \mathbf{0})$
$1 \leq i \leq k$	$l_i \Rightarrow \left(\lvert[\phi]\rvert_{k+1} \Leftrightarrow \lvert[\phi]\rvert_i\right)$

Encoding LTL Operators (1/4)

- Encoding propositional operators is straighforward

	φ	constraint
$0 \leq i \leq k$	p	$\lvert[p]\rvert_i \Leftrightarrow p_i$
	$\neg p$	$\lvert[\neg p]\rvert_i \Leftrightarrow \neg p_i$
	$\psi_1 \wedge \psi_2$	$\lvert[\psi_1 \wedge \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_i \wedge \lvert[\psi_2]\rvert_i$
	$\psi_1 \vee \psi_2$	$\lvert[\psi_1 \vee \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_i \vee \lvert[\psi_2]\rvert_i$

Encoding LTL Operators (2/4)

- Basic (but incomplete) translation of temporal operators follows the standard recursive definitions
- Is not alone correct for (k,l)-loop cases

	φ	encoding
$0 \leq i \leq k$	$\mathbf{X}\,\phi$	$\lvert[\mathbf{X}\,\phi]\rvert_i \Leftrightarrow \lvert[\phi]\rvert_{i+1}$
	$\mathbf{F}\,\phi$	$\lvert[\mathbf{F}\,\phi]\rvert_i \Leftrightarrow \lvert[\phi]\rvert_i \vee \lvert[\mathbf{F}\,\phi]\rvert_{i+1}$
	$\mathbf{G}\,\phi$	$\lvert[\mathbf{G}\,\phi]\rvert_i \Leftrightarrow \lvert[\phi]\rvert_i \wedge \lvert[\mathbf{G}\,\phi]\rvert_{i+1}$
	$\psi_1\,\mathbf{U}\,\psi_2$	$\lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_i \Leftrightarrow \lvert[\psi_2]\rvert_i \vee \left(\lvert[\psi_1]\rvert_i \wedge \lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_{i+1}\right)$
	$\psi_1\,\mathbf{R}\,\psi_2$	$\lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_i \Leftrightarrow \lvert[\psi_2]\rvert_i \wedge \left(\lvert[\psi_1]\rvert_i \vee \lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_{i+1}\right)$

Illustration of the Encoding

- Mutex example, $k = 3$, no loop
- The no loop case: correct translation

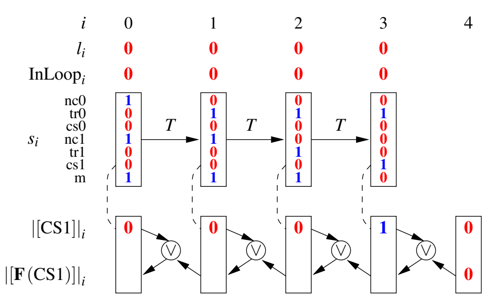


Illustration of the Encoding

- Mutex example, $k = 3$, no loop
- The no loop case: correct translation

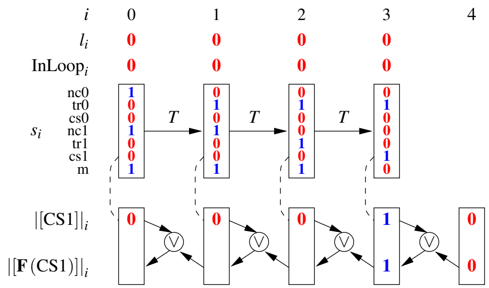


Illustration of the Encoding

- Mutex example, $k = 3$, no loop
- The no loop case: correct translation

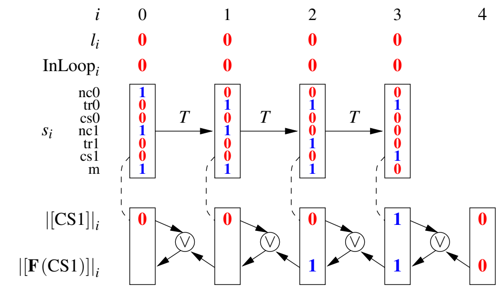


Illustration of the Encoding

- Mutex example, $k = 3$, no loop
- The no loop case: correct translation

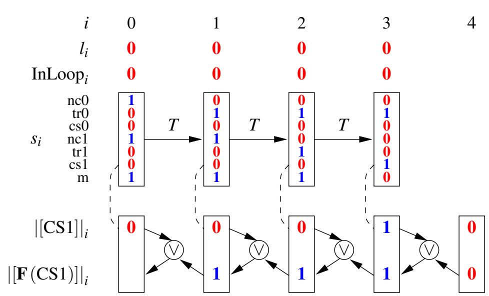


Illustration of the Encoding

- Mutex example, $k = 3$, no loop
- The no loop case: correct translation

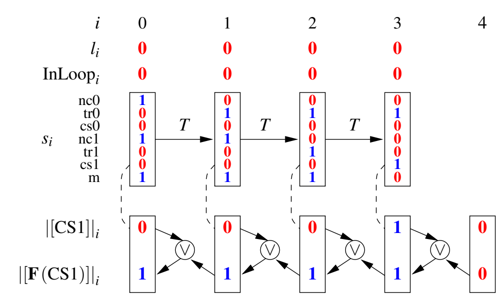


Illustration of the Encoding

- Mutex example, $k = 3$, no loop
- The no loop case: correct translation

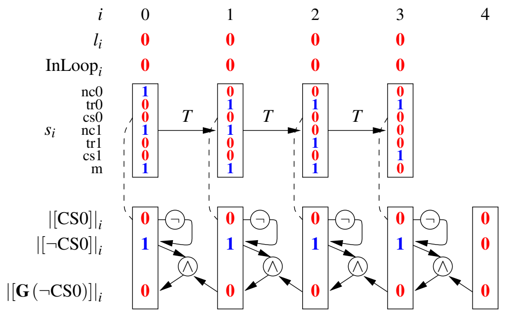


Illustration of the Encoding

- Mutex example, $k = 4$, $l_2 = \mathbf{1}$
- The (k,l)-loop case: incorrect evaluation of $\mathbf{F}(\mathrm{CS0})$ at time points $0 \leq i \leq 4$

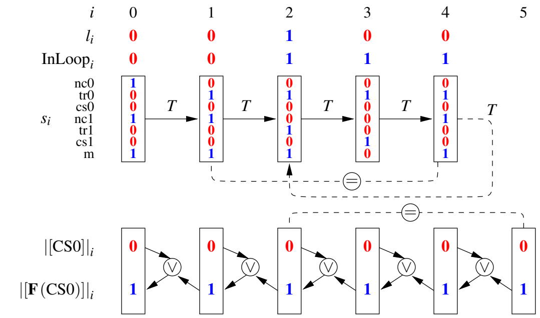
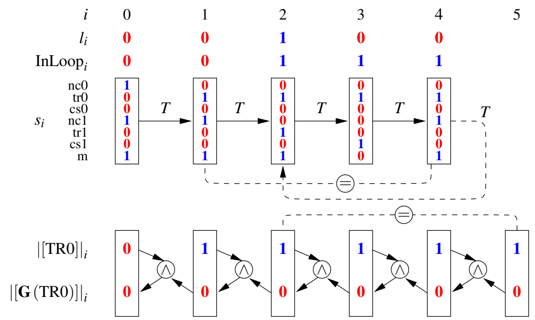


Illustration of the Encoding

- Mutex example, $k = 4$, $l_2 = \mathbf{1}$
- The (k,l)-loop case: incorrect evaluation of $\mathbf{G}(\mathrm{TR0})$ at time points $1 \leq i \leq 4$



Encoding LTL Operators (3/4)

- The (k,l)-loop cases require an auxiliary encoding to force the cyclic dependencies to evaluate correctly
- Idea: $\langle\langle \mathbf{F}\,\phi \rangle\rangle_k$ evaluates to true iff ϕ evaluates to true at least once in the selected loop
- Idea: $\langle\langle \mathbf{G}\,\phi \rangle\rangle_k$ evaluates to true iff ϕ evaluates to true in all states in the selected loop

Base	$\langle\langle \mathbf{F}\,\phi \rangle\rangle_0 \Leftrightarrow \mathbf{0}$ $\langle\langle \mathbf{G}\,\phi \rangle\rangle_0 \Leftrightarrow \mathbf{1}$
$1 \leq i \leq k$	$\langle\langle \mathbf{F}\,\phi \rangle\rangle_i \Leftrightarrow \langle\langle \mathbf{F}\,\phi \rangle\rangle_{i-1} \vee (\mathrm{InLoop}_i \wedge \lvert[\phi]\rvert_i)$ $\langle\langle \mathbf{G}\,\phi \rangle\rangle_i \Leftrightarrow \langle\langle \mathbf{G}\,\phi \rangle\rangle_{i-1} \wedge \neg(\mathrm{InLoop}_i \wedge \neg\lvert[\phi]\rvert_i)$

Encoding LTL Operators (4/4)

- Force cyclic dependencies to evaluate correctly

ϕ	Added constraint
$\mathbf{F}\,\psi_1$	LoopExists $\Rightarrow (\lvert[\mathbf{F}\,\psi_1]\rvert_k \Rightarrow \langle\langle \mathbf{F}\,\psi_1 \rangle\rangle_k)$
$\mathbf{G}\,\psi_1$	LoopExists $\Rightarrow (\lvert[\mathbf{G}\,\psi_1]\rvert_k \Leftarrow \langle\langle \mathbf{G}\,\psi_1 \rangle\rangle_k)$
$\psi_1\,\mathbf{U}\,\psi_2$	LoopExists $\Rightarrow (\lvert[\psi_1\,\mathbf{U}\,\psi_2]\rvert_k \Rightarrow \langle\langle \mathbf{F}\,\psi_2 \rangle\rangle_k)$
$\psi_1\,\mathbf{R}\,\psi_2$	LoopExists $\Rightarrow (\lvert[\psi_1\,\mathbf{R}\,\psi_2]\rvert_k \Leftarrow \langle\langle \mathbf{G}\,\psi_2 \rangle\rangle_k)$

Illustration of the Encoding

- Mutex example, $k = 4$, $l_2 = \mathbf{1}$
- Correctly evaluates $\lvert[\mathbf{F}\,(\mathrm{CS0})]\rvert_i$ to $\mathbf{0}$ for $0 \le i \le k$

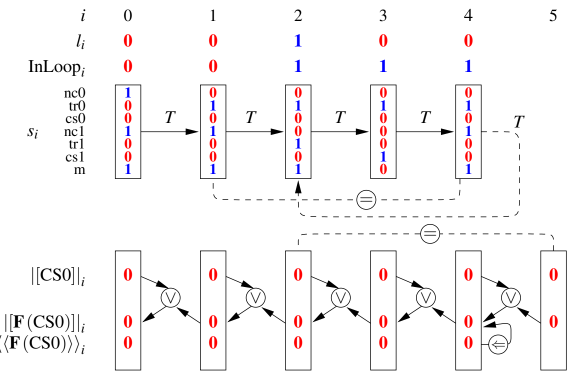
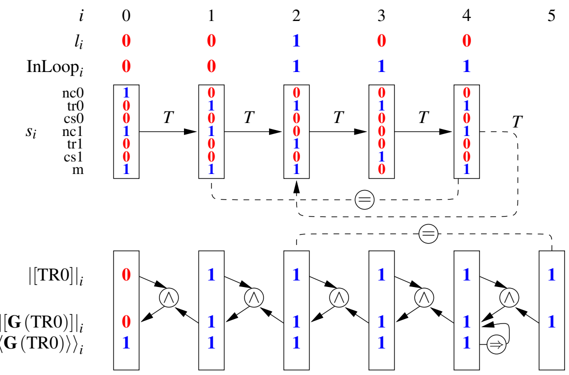


Illustration of the Encoding

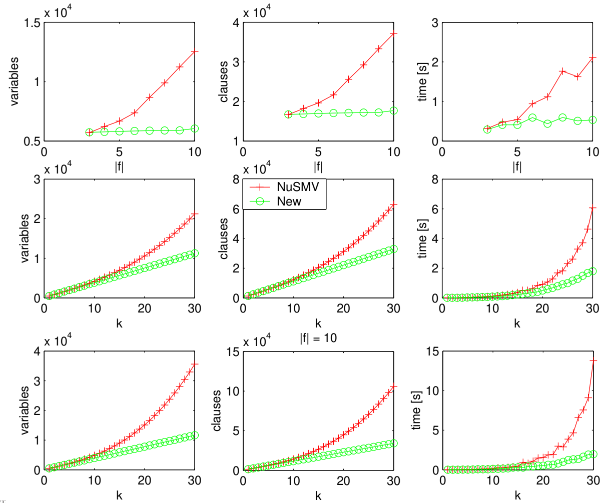
- Mutex example, $k = 4$, $l_2 = \mathbf{1}$
- Correctly evaluates $\lvert[\mathbf{G}\,(\mathrm{TR0})]\rvert_i$ to $\mathbf{1}$ for $1 \le i \le k$



Some Experimental Results

- From our FMCAD'04 paper.
- Encoding similar to (but not exactly same as) the one presented here
- Random formulae on small random Kripke structures.
- Formula sizes between 3–10, and k from 0–30.
- A few real-life examples.
- Compare with encoding of NuSMV 2.1.
- Measure: number of variables and clauses in the CNF encoding, time to solve instance.

Benchmarks I



Benchmarks II

Model	k	NuSMV			New		
		vars	*clauses*	*time*	*vars*	*clauses*	*time*
abp	16	19,476	57,373	32.3	18,024	52,969	7.4
brp	10	7,599	21,811	1.3	7,471	21,397	1.5
	15	11,494	33,226	18.7	11,116	32,047	17.9
	20	15,514	45,016	471	14,761	42,697	484
dme	10	53,400	141,438	2.0	53,293	141,087	2.6
	20	104,885	283,733	180	104,173	281,537	471
	30	156,870	427,528	1,199	155,053	421,987	1,544
pci	10	56,414	167,753	58.3	55,911	166,214	51.5
	15	85,359	254,133	568	83,756	249,279	382
	20	115,204	343,213	5,921	111,601	332,344	2,102
srg16	20	N/A	N/A	N/A	5,196	14,921	2.7
	40	N/A	N/A	N/A	10,336	29,841	22.3
	60	N/A	N/A	N/A	15,476	44,761	83.0

BMC and Incremental SAT Solving

BMC and Incremental SAT Solving

- SAT problems from BMC with increasing bounds are quite similar:
  $|[M,\psi,0]| \precsim |[M,\psi,1]| \precsim |[M,\psi,2]| \precsim \dots$
- State-of-the-art propositional SAT solvers such as zChaff and MiniSat can exploit this
  - The learned conflict clauses based on the part of the SAT instance that stays the same can be transferred to the next instance

Incremental SAT Solving

- For instance, zChaff provides the following interface:
  - int SAT_AddVariable(SAT_Manager mng)
    Introduce a new Boolean variable
  - void SAT_AddClause(SAT_Manager, int *lits, int num_lits, int gid)
    Add the clause to group gid ($0 \leq$ gid ≤ 31)
  - int SAT_Solve(SAT_Manager mng)
    Solve the current instance
  - void SAT_DeleteClauseGroup(SAT_Manager, int gid)
    Delete all clauses in group gid and all learned clauses depending on them

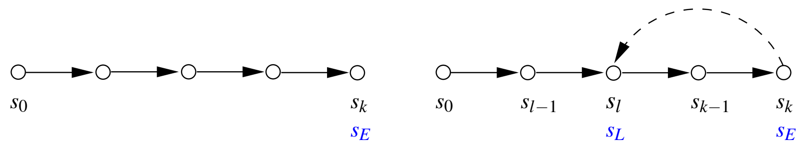
Basic Approach to Incrementality

- Divide the BMC encoding into three parts:
  - Base encoding α - stays the same for all bounds
  - k-invariant part β_i - is independent of the actual value of the bound k
  - k-dependent part γ_i - is dependent on the value of the bound k
- Example of increasing bound from 3 to 4:
  - $\alpha \wedge \beta_0 \wedge \beta_1 \wedge \beta_2 \wedge \gamma_2$
  - $\alpha \wedge \beta_0 \wedge \beta_1 \wedge \beta_2 \wedge \beta_3 \wedge \gamma_3$

Incrementality

- Provide an incremental SAT interface which drops k-dependent parts when bound is increased
- The underlying incremental SAT-solver
  - can reuse everything learned from the base and k-invariant parts
  - has to drop everything learned from the k-dependent part
- Goal: minimise the size of the k-dependent part

Incrementality: Proxy States



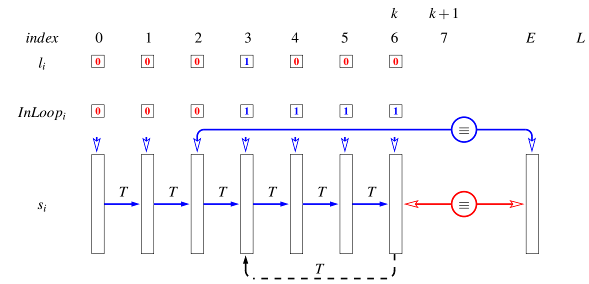
(a) no loop (b) (k,l)-loop

- Main idea: Introduce proxy states s_E (the "end state") and s_L (the "loop state", successor of s_E)
- Use s_E and s_L in constraints instead of s_k and s_{k+1}
- In k-dependent part force s_E (s_L) equivalent to s_k (s_{k+1})

Incrementality: Loop Constraints

	$\lvert[\text{LoopConstraints}]\rvert_k$
Base	$l_0 \Leftrightarrow \mathbf{0}$
	$\text{InLoop}_0 \Leftrightarrow \mathbf{0}$
	$l_i \Rightarrow (s_{i-1} = s_E)$
$k-$invariant	$\text{InLoop}_i \Leftrightarrow \text{InLoop}_{i-1} \vee l_i,$
$1 \leq i \leq k$	$l_i \Rightarrow \neg\text{InLoop}_{i-1}$
$k-$dependent	$s_E = s_k$
	$\text{LoopExists} \Leftrightarrow \text{InLoop}_k$

Illustration of the Encoding, $l_3 = \mathbf{1}$

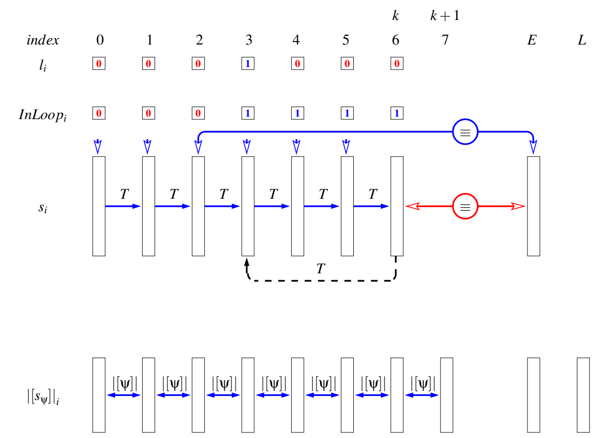


Incrementality: Encoding LTL Operators

- For each subformula φ of ψ, introduce a variable $\lvert[\varphi]\rvert_i$ where $i \in \{0, 1, \ldots, k+1, L, E\}$

	φ	encoding
	p	$\lvert[p]\rvert_i \Leftrightarrow p_i$
	$\neg p$	$\lvert[\neg p]\rvert_i \Leftrightarrow \neg p_i$
k-invariant	$\psi_1 \wedge \psi_2$	$\lvert[\psi_1 \wedge \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_i \wedge \lvert[\psi_2]\rvert_i$
$1 \leq i \leq k$	$\psi_1 \vee \psi_2$	$\lvert[\psi_1 \vee \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_i \vee \lvert[\psi_2]\rvert_i$
	$\mathbf{X}\phi$	$\lvert[\mathbf{X}\phi]\rvert_i \Leftrightarrow \lvert[\phi]\rvert_{i+1}$
	...	...

Illustration of the Encoding, $l_3 = \mathbf{1}$

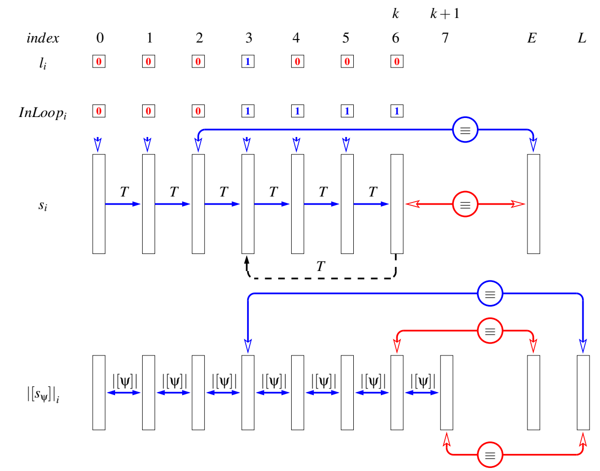


Incrementality: Last State Constraints

- Force loop constraints also in the LTL encoding part

	$\|[\text{LastStateConstraints}]\|_k$
Base	$\neg \mathit{LoopExists} \Rightarrow (\|[\varphi]\|_L \Leftrightarrow \mathbf{0})$
k-invariant	$l_i \Rightarrow (\|[\varphi]\|_L \Leftrightarrow \|[\varphi]\|_i)$
k-dependent	$\|[\varphi]\|_E \Leftrightarrow \|[\varphi]\|_k$
	$\|[\varphi]\|_{k+1} \Leftrightarrow \|[\varphi]\|_L$

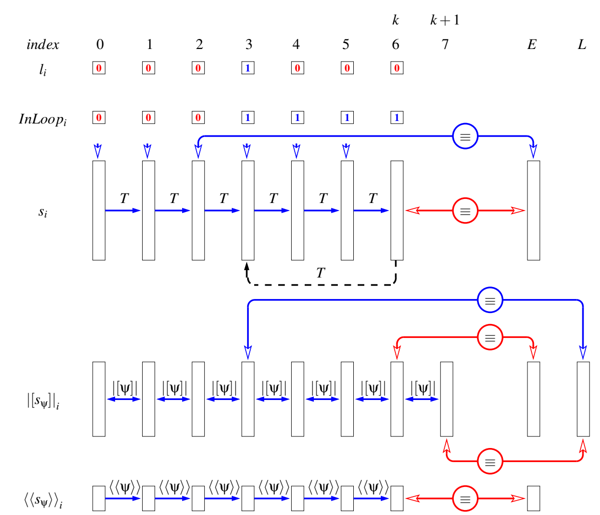
Illustration of the Encoding, $l_3 = 1$



Incrementality: Auxiliary Translation

Base	$\langle\langle \mathbf{F}\,\phi \rangle\rangle_0 \Leftrightarrow \mathbf{0}$
	$\langle\langle \mathbf{G}\,\phi \rangle\rangle_0 \Leftrightarrow \mathbf{1}$
	$\text{LoopExists} \Rightarrow (\|[\mathbf{F}\,\psi_1]\|_E \Rightarrow \langle\langle \mathbf{F}\,\psi_1 \rangle\rangle_E)$
	$\text{LoopExists} \Rightarrow (\|[\mathbf{G}\,\psi_1]\|_E \Leftarrow \langle\langle \mathbf{G}\,\psi_1 \rangle\rangle_E)$
	$\text{LoopExists} \Rightarrow (\|[\psi_1\,\mathbf{U}\,\psi_2]\|_E \Rightarrow \langle\langle \mathbf{F}\,\psi_2 \rangle\rangle_E)$
	$\text{LoopExists} \Rightarrow (\|[\psi_1\,\mathbf{R}\,\psi_2]\|_E \Leftarrow \langle\langle \mathbf{G}\,\psi_2 \rangle\rangle_E)$
k-invariant	$\langle\langle \mathbf{F}\,\phi \rangle\rangle_i \Leftrightarrow \langle\langle \mathbf{F}\,\phi \rangle\rangle_{i-1} \lor (\text{InLoop}_i \land \|[\phi]\|_i)$
$1 \le i \le k$	$\langle\langle \mathbf{G}\,\phi \rangle\rangle_i \Leftrightarrow \langle\langle \mathbf{G}\,\phi \rangle\rangle_{i-1} \land \neg(\text{InLoop}_i \land \neg\|[\phi]\|_i)$
k-dependent	$\langle\langle \mathbf{F}\,\phi \rangle\rangle_E \Leftrightarrow \langle\langle \mathbf{F}\,\phi \rangle\rangle_k$
	$\langle\langle \mathbf{G}\,\phi \rangle\rangle_E \Leftrightarrow \langle\langle \mathbf{G}\,\phi \rangle\rangle_k$

Illustration of the Encoding, $l_3 = 1$



Incrementality: Experimental Results

- From our CAV'05 paper
- The VMCAI benchmarks have non-trivial LTL (with past operators) properties
- The IBM benchmarks have simple invariant properties
- 1 hour time and 900MB memory limits
- k columns denote the bound reached within the limits
- Conclusion: incrementality usually gives a nice performance boost

Experiments, part 1

	NuSMV 2.2.3			New incremental			New non-inc.		
problem	t/f	k	time	t/f	k	time	t/f	k	time
VMCAI2005/abp4	f	16	70	f	16	56	f	16	55
VMCAI2005/brp		28			1771			166	
VMCAI2005/dme4		23			56			51	
VMCAI2005/pci		15		f	18	2388		17	
VMCAI2005/srg5		12			736			210	

- Best
- Worst

Experiments, part 2

	NuSMV 2.2.3			New incremental.			New non-inc.		
problem	t/f	k	time	t/f	k	time	t/f	k	time
IBM/IBM_FV_2002_01	f	14	90	f	14	44	f	14	87
IBM/IBM_FV_2002_03	f	32	134	f	32	32	f	32	200
IBM/IBM_FV_2002_04	f	24	38	f	24	12	f	24	90
IBM/IBM_FV_2002_05	f	31	258	f	31	17	f	31	251
IBM/IBM_FV_2002_06	f	31	573	f	31	77	f	31	723
IBM/IBM_FV_2002_09		232			787			81	
IBM/IBM_FV_2002_15	f	9	38	f	9	3	f	9	4
IBM/IBM_FV_2002_18		26		f	29	2362		26	
IBM/IBM_FV_2002_19	f	29	3057	f	29	86		28	
IBM/IBM_FV_2002_20		27			35			26	
IBM/IBM_FV_2002_21	f	29	2276	f	29	144	f	29	2741
IBM/IBM_FV_2002_22		25			49			25	
IBM/IBM_FV_2002_23		25			31			24	
IBM/IBM_FV_2002_27	f	25	298	f	25	15	f	25	322
IBM/IBM_FV_2002_28	f	14	1046	f	14	245	f	14	1023
IBM/IBM_FV_2002_29		14			17			14	

Incrementality: Closely Related Work

- Eén, N. and Sörensson N.: Temporal Induction by Incremental SAT Solving. BMC'03.
  - An incremental and complete BMC procedure for invariants.
- Benedetti,M. and Bernardini, S.: Incremental compilation-to-SAT procedures. SAT'04.
  - Incremental version of Benedetti-Cimatti translation for PLTL

Making BMC for LTL Complete

Making BMC for LTL Complete

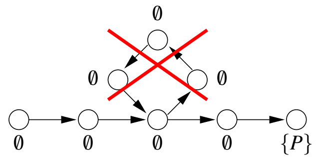
- Goal: a procedure that can also prove that $M \models \psi$, not only find counter-examples
- There is always a k such that $M \models \psi$ iff $M \models_k \psi$ but such k can be
  - large: $O(|S| \cdot |\psi| \cdot 2^{|\psi|})$, and
  - hard to deduce.

Making BMC for LTL Complete

- Our approach is based on a forward variant of temporal induction (aka k-induction) for proving invariant properties
  - Sheeran, M., Singh, S., and Stålmarck, G.: Checking Safety Properties Using Induction and a SAT-Solver. FMCAD'00.
    - A complete BMC procedure for invariants.
  - Eén, N. and Sörensson N.: Temporal Induction by Incremental SAT Solving. BMC'03.
    - An incremental and complete BMC procedure for invariants.

A Simple Forward Variant of Temporal Induction

- Goal: Check whether $M \models \mathbf{G}(P)$, where $P \in AP$
- Completeness is achieved by only considering simple (loop-free) bounded paths
- If there are no simple bounded paths of lenght k' or greater and $M \not\models_k \mathbf{F}(\neg P)$ for all $k < k'$, then $M \models \mathbf{G}(P)$



A Simple Forward Variant of Temporal Induction

- The simple path constraint can be expressed as $|[SimplePath]|_k := \bigwedge_{0 \le i < j \le k} \neg(s_i = s_j)$
- Is of size $O(k^2)$
- A sorting network approach can be used instead, allowing the simple path constraint to be expressed in size $O(k \log^2 k)$ or $O(k \log k)$:
  - Kroening, D., and Strichman, O.: Efficient Computation of Recurrence Diameters. VMCAI'03.

A Simple Forward Variant of Temporal Induction

- Procedure: start with $k = 0$
  - If $|[M]|_k \wedge |[SimplePath]|_k$ is unsatisfiable, return "$M \models \mathbf{G}(P)$ holds"
  - If $|[M]|_k \wedge |[SimplePath]|_k \wedge |[\neg P]|_k$ is satisfiable, return "$M \models \mathbf{G}(P)$ does not hold"
  - Else set $k = k + 1$ and restart

A Simple Forward Variant of Temporal Induction

- Illustration: Check whether $M \models \mathbf{G}(P)$
  - $I(s_0)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$

A Simple Forward Variant of Temporal Induction

- Illustration: Check whether $M \models \mathbf{G}(P)$
  - $I(s_0)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge \neg P_0$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science Advanced Tutorial on Bounded Model Checking at ACSD'06 - ATPN'06, Keijo Heljanko and Tommi Junttila – 79/130

A Simple Forward Variant of Temporal Induction

- Illustration: Check whether $M \models \mathbf{G}(P)$
  - $I(s_0)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge \neg P_0$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge (s_0 \neq s_1)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$

A Simple Forward Variant of Temporal Induction

- Illustration: Check whether $M \models \mathbf{G}(P)$
  - $I(s_0)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge \neg P_0$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge (s_0 \neq s_1)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge (s_0 \neq s_1) \wedge \neg P_1$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science
Advanced Tutorial on Bounded Model Checking at ACSD'06 - ATPN'06, Keijo Heljanko and Tommi Junttila – 79/130

A Simple Forward Variant of Temporal Induction

- Illustration: Check whether $M \models \mathbf{G}(P)$
  - $I(s_0)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge \neg P_0$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge (s_0 \neq s_1)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge (s_0 \neq s_1) \wedge \neg P_1$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge (s_0 \neq s_1) \wedge (s_0 \neq s2) \wedge (s_1 \neq s_2)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science
Advanced Tutorial on Bounded Model Checking at ACSD'06 - ATPN'06, Keijo Heljanko and Tommi Junttila – 79/130

A Simple Forward Variant of Temporal Induction

- Illustration: Check whether $M \models \mathbf{G}(P)$
  - $I(s_0)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge \neg P_0$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge (s_0 \neq s_1)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge (s_0 \neq s_1) \wedge \neg P_1$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge (s_0 \neq s_1) \wedge (s_0 \neq s2) \wedge (s_1 \neq s_2)$ is unsat $\Rightarrow$ return $M \models \mathbf{G}(P)$
  - $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge (s_0 \neq s_1) \wedge (s_0 \neq s2) \wedge (s_1 \neq s_2) \wedge \neg P_2$ is sat $\Rightarrow$ return $M \not\models \mathbf{G}(P)$
  - ...

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science
Advanced Tutorial on Bounded Model Checking at ACSD'06 - ATPN'06, Keijo Heljanko and Tommi Junttila – 79/130

Making BMC for LTL Complete

- Extend temporal induction to LTL BMC
- A *minimal stem* is a bounded path which is a prefix of some minimal-length bounded witness path
- Given a bound k, create two formulas:
  - completeness formula:
    (1) satisfiable for all minimal stems, and
    (2) there is a $k \geq 0$ such that the completeness formula becomes unsatisfiable.
  - witness formula that is satisfiable iff there is a witness with bound k: this is simply $|[M, \psi, k]|$.

Procedure

- Similar to the temporal induction procedure above
- Start with bound $k = 0$.
  1. If the completeness formula for k is unsatisfiable then there are no minimal-length witnesses to ψ for any $k' \geq k$. Return "$M \models \psi$".
  2. If the witness formula for k is satisfiable then there is a witness to ψ. Return "$M \not\models \psi$".
  3. Otherwise, increment bound k by one and repeat.

Completeness Formula, part (1)

- Completeness formula, requirement (1):
  (1) satisfiable for all minimal stems.
- To satisfy (1):
  - Use as part (1) of the completeness formula: the witness formula with all k-dependent constraints removed.
- The future state $k+1$ is now totally free
- Example:

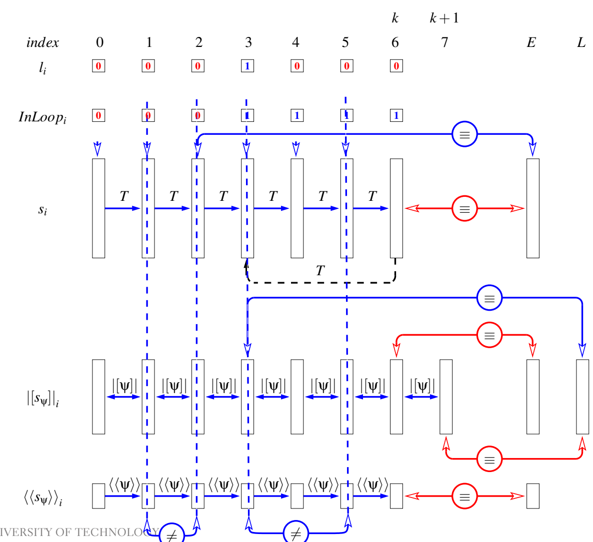
Completeness, $k = 3$:	$\alpha \wedge \beta_0 \wedge \beta_1 \wedge \beta_2$
Witness, $k = 3$:	$\alpha \wedge \beta_0 \wedge \beta_1 \wedge \beta_2 \wedge \gamma_2$
Completeness, $k = 4$:	$\alpha \wedge \beta_0 \wedge \beta_1 \wedge \beta_2 \wedge \beta_3$
Witness, $k = 4$:	$\alpha \wedge \beta_0 \wedge \beta_1 \wedge \beta_2 \wedge \beta_3 \wedge \gamma_3$

Completeness Formula, part (2)

- Completeness formula, requirement (2):
  (2) there exists a $k \geq 0$ such that the completeness formula is unsatisfiable.
- Solution: Conjunct the part (1) of the completeness formula with a $|[SimplePath]|_k$ formula which is: (a) satisfiable for all minimal stems but (b) unsatisfiable for bounded witnesses of non-minimal length.

$$|[SimplePath]|_k := \bigwedge_{0 \leq i < j \leq k} \neg \left(\begin{array}{ll} s_i = s_j & \wedge \\ |[s_\psi]|_i = |[s_\psi]|_j & \wedge \\ \langle\langle s_\psi \rangle\rangle_i = \langle\langle s_\psi \rangle\rangle_j & \wedge \\ \mathsf{InLoop}_i = \mathsf{InLoop}_j \end{array} \right)$$

Illustration of the Encoding, $l_3 = 1$



Compatibility

- The presented completeness approach
  - is compatible with the incremental SAT solving approach presented earlier
  - can be extended to the PLTL approach presented later in a straightforward way

Completeness: Some Experiments

- From our CAV'05 paper
- For some simple models, properties can be proven to hold
- The completeness check can sometimes incur a considerable performance penalty
- Using incrementality can compensate this

Experiments, part 1

	NuSMV 2.2.3			New incremental			New non-inc.			New incremental with completeness			New non-inc. with completeness		
problem	t/f	k	time	t/f	k	time	t/f	k	time	t/f	k	time	t/f	k	time
VMCAI2005/abp4	f	16	70	f	16	56	f	16	55	f	16	26	f	16	68
VMCAI2005/brp		28			1771			166			89			39	
VMCAI2005/dme4		23			56			51			57			39	
VMCAI2005/pci		15		f	18	2388		17			18			17	
VMCAI2005/srg5		12			736			210			54			44	
bmc/barrel5		28			67			26		t	11	63	t	11	314
ctl-ltl/counter_1		181			8025			3019		t	24	0	t	24	0
ctl-ltl/mutex		196			6855			2578		t	19	0	t	19	0
ctl-ltl/periodic		561			2063			781		t	201	593	t	201	2596
ctl-ltl/ring		159			713			165		t	66	3203		44	
ctl-ltl/short		182			2496			800		t	11	0	t	11	0

- Best
- Worst

Experiments, part 2

	NuSMV 2.2.3			New incremental.			New non-inc.			New incremental with completeness			New non-inc. with completeness		
problem	t/f	k	time	t/f	k	time	t/f	k	time	t/f	k	time	t/f	k	time
IBM/IBM_FV_2002_01	f	14	90	f	14	44	f	14	87	f	14	54	f	14	113
IBM/IBM_FV_2002_03	f	32	134	f	32	32	f	32	200	f	32	74	f	32	727
IBM/IBM_FV_2002_04	f	24	38	f	24	12	f	24	90	f	24	38	f	24	156
IBM/IBM_FV_2002_05	f	31	258	f	31	17	f	31	251	f	31	52	f	31	617
IBM/IBM_FV_2002_06	f	31	573	f	31	77	f	31	723	f	31	270	f	31	2032
IBM/IBM_FV_2002_09		232			787			81			81			76	
IBM/IBM_FV_2002_15	f	9	38	f	9	3	f	9	4	f	9	3	f	9	9
IBM/IBM_FV_2002_18		26		f	29	2362		26		f	29	1789		24	
IBM/IBM_FV_2002_19	f	29	3057	f	29	86		28		f	29	300		23	
IBM/IBM_FV_2002_20		27			35			26			35			24	
IBM/IBM_FV_2002_21	f	29	2276	f	29	144	f	29	2741	f	29	239		24	
IBM/IBM_FV_2002_22		25			49			25			42			20	
IBM/IBM_FV_2002_23		25			31			24			31			21	
IBM/IBM_FV_2002_27	f	25	298	f	25	15	f	25	322	f	25	44	f	25	406
IBM/IBM_FV_2002_28	f	14	1046	f	14	245	f	14	1023	f	14	278	f	14	1160
IBM/IBM_FV_2002_29		14			17			14			20			14	

Completeness: Closely Related Work

- Sheeran, M., Singh, S., and Stålmarck, G.: Checking Safety Properties Using Induction and a SAT-Solver. FMCAD'00.
  - A complete BMC procedure for invariants.
- Eén, N. and Sörensson N.: Temporal Induction by Incremental SAT Solving. BMC'03.
  - An incremental and complete BMC procedure for invariants.

Completeness: Closely Related Work

- Clarke, E. M., Kroening, D., Ouaknine, J., and Strichman, O.: Completeness and Complexity of Bounded Model Checking. VMCAI'04.
  - Complete BMC for LTL using (non-generalised) Büchi automata.
- Awedh, M. and Somenzi, F.: Proving More Properties with Bounded Model Checking. CAV'04.
  - Complete BMC for LTL using (non-generalised) Büchi automata.

Completeness: Other Related Work

- Abdulla, P.A., Bjesse, B., and Eén, N.: Symbolic Reachability Analysis Based on SAT-solvers. TACAS'00.
  - A SAT-based procedure for checking invariants by using explicit quantifier elimination
- McMillan, K.L.,: Interpolation and SAT-Based Model Checking. CAV'03.
  - A complete SAT-based procedure for checking invariants by using Craig interpolants

BMC for LTL with Past Operators

LTL with Past: PLTL

- Extends LTL with past operators

$\mathbf{Y}\,\psi_1$	"in previous state" (false in the beginning)
$\mathbf{Z}\,\psi_1$	"in previous state" (true in the beginning)
$\mathbf{O}\,\psi_1$	"once"
$\mathbf{H}\,\psi_1$	"historically"
$\psi_1\,\mathbf{S}\,\psi_2$	"since"
$\psi_1\,\mathbf{T}\,\psi_2$	"trigger"

LTL with Past: PLTL

- 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 $(req\,\mathbf{R}\,\neg ack) \wedge \mathbf{G}\,(ack \Rightarrow \mathbf{X}\,(req\,\mathbf{R}\,\neg ack))$.
- Although model checking PSPACE complete as for LTL, more complicated algorithms in practice.

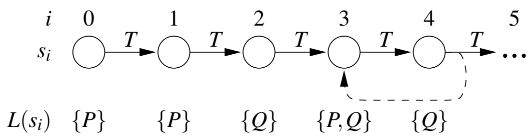
Semantics of PLTL: Past formulas

- Let $\pi = s_0 s_1 \ldots$ be an infinite sequence of states with labelling $L(s_i) \in 2^{AP}$.

$$
\begin{array}{lll}
\pi^i \models \mathbf{Y}\psi & \Leftrightarrow & (i > 0) \wedge (\pi^{i-1} \models \psi) \\
\pi^i \models \mathbf{Z}\psi & \Leftrightarrow & (i = 0) \vee (\pi^{i-1} \models \psi) \\
\pi^i \models \mathbf{O}\,\psi_1 & \Leftrightarrow & \exists 0 \le j \le i : \pi^j \models \psi_1 \\
\pi^i \models \mathbf{H}\,\psi_1 & \Leftrightarrow & \forall 0 \le j \le i : \pi^j \models \psi_1 \\
\pi^i \models \psi_1\,\mathbf{S}\,\psi_2 & \Leftrightarrow & \exists 0 \le j \le i : (\pi^j \models \psi_2 \wedge \forall j < n \le i : \pi^n \models \psi_1) \\
\pi^i \models \psi_1\,\mathbf{T}\,\psi_2 & \Leftrightarrow & (\forall 0 \le j \le i : \pi^j \models \psi_2) \vee \\
 & & (\exists 0 \le j \le i : \pi^j \models \psi_1 \wedge \forall j \le n \le i : \pi^n \models \psi_2)
\end{array}
$$

- Bounded semantics: replace $\models$ with $\models_{nl}$

Semantics of PLTL



- $\pi^0 \models \mathbf{O}(P)$, $\pi^4 \models \mathbf{O}(P)$
- $\pi^1 \models \mathbf{H}(P)$, $\pi^2 \not\models \mathbf{H}(P)$
- $\pi^0 \not\models \mathbf{Y}(Q)$, $\pi^0 \models \mathbf{Z}(Q)$, $\pi^2 \models \mathbf{Y}(P)$
- $\pi^2 \models Q\,\mathbf{S}\,P$
- $\pi^2 \not\models P\,\mathbf{T}\,Q$, $\pi^4 \models P\,\mathbf{T}\,Q$

Positive Normal Form for PLTL

- Apply equations:

$$\begin{aligned}
\neg(\mathbf{Y}\,\psi) &\equiv \mathbf{Z}\,(\neg\psi)\\
\neg(\mathbf{Z}\,\psi) &\equiv \mathbf{Y}\,(\neg\psi)\\
\neg(\mathbf{O}\,\psi) &\equiv \mathbf{H}\,(\neg\psi)\\
\neg(\mathbf{H}\,\psi) &\equiv \mathbf{O}\,(\neg\psi)\\
\neg(\psi_1\,\mathbf{S}\,\psi_2) &\equiv (\neg\psi_1)\,\mathbf{T}\,(\neg\psi_2)\\
\neg(\psi_1\,\mathbf{T}\,\psi_2) &\equiv (\neg\psi_1)\,\mathbf{S}\,(\neg\psi_2)
\end{aligned}$$

Encoding I: No Virtual Unrolling

- Augment our LTL encoding $|[M,\psi,k]|$ with past operator encodings
- Is of same compact size: $O(|I| + k\cdot|T| + k\cdot|\psi|)$
- Cannot always detect minimal counter-examples:
  - there are formulas ψ such that $|[M,\psi,k]|$ evaluates to false although there is a (k,l)-loop π in M such that $\pi \models_k \psi$
  - however, in such cases, there is a $k' > k$ such that π is a (k',l')-loop and $|[M,\psi,k']|$ evaluates to true

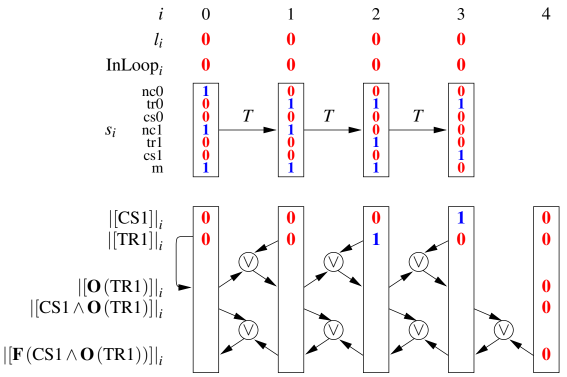
No Virtual Unrolling Encoding: Past Operators

- Basic (but incomplete) translation follows the standard recursive definitions
- Is not alone correct for (k,l)-loops

	$i=0$	$1 \le i \le k$
$\lvert[\mathbf{Y}\,\psi_1]\rvert_i$	$\mathbf{0}$	$\lvert[\psi_1]\rvert_{i-1}$
$\lvert[\mathbf{Z}\,\psi_1]\rvert_i$	$\mathbf{1}$	$\lvert[\psi_1]\rvert_{i-1}$
$\lvert[\mathbf{O}\,\psi_1]\rvert_i$	$\lvert[\psi_1]\rvert_i$	$\lvert[\psi_1]\rvert_i \vee \lvert[\mathbf{O}\,\psi_1]\rvert_{i-1}$
$\lvert[\mathbf{H}\,\psi_1]\rvert_i$	$\lvert[\psi_1]\rvert_i$	$\lvert[\psi_1]\rvert_i \wedge \lvert[\mathbf{H}\,\psi_1]\rvert_{i-1}$
$\lvert[\psi_1\,\mathbf{S}\,\psi_2]\rvert_i$	$\lvert[\psi_2]\rvert_i$	$\lvert[\psi_2]\rvert_i \vee \left(\lvert[\psi_1]\rvert_i \wedge \lvert[\psi_1\,\mathbf{S}\,\psi_2]\rvert_{i-1}\right)$
$\lvert[\psi_1\,\mathbf{T}\,\psi_2]\rvert_i$	$\lvert[\psi_2]\rvert_i$	$\lvert[\psi_2]\rvert_i \wedge \left(\lvert[\psi_1]\rvert_i \vee \lvert[\psi_1\,\mathbf{T}\,\psi_2]\rvert_{i-1}\right)$

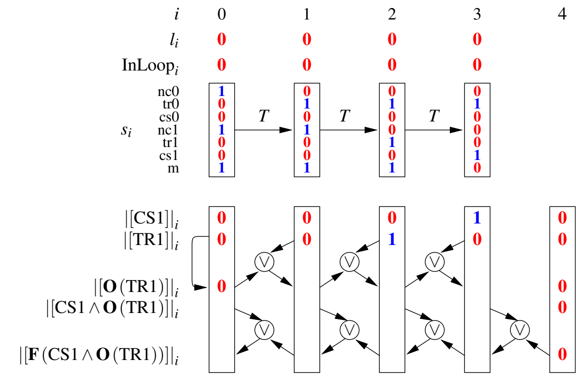
No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



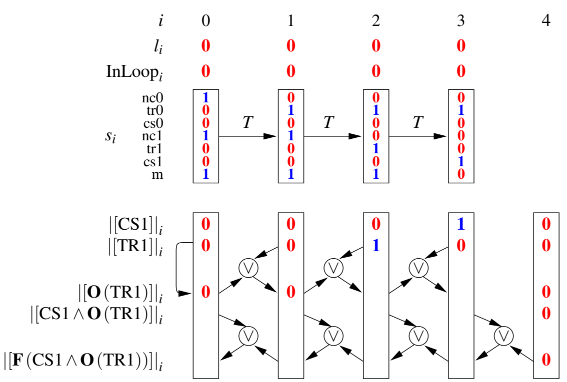
No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



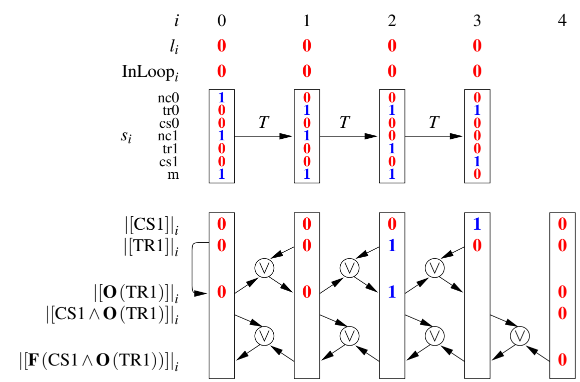
No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



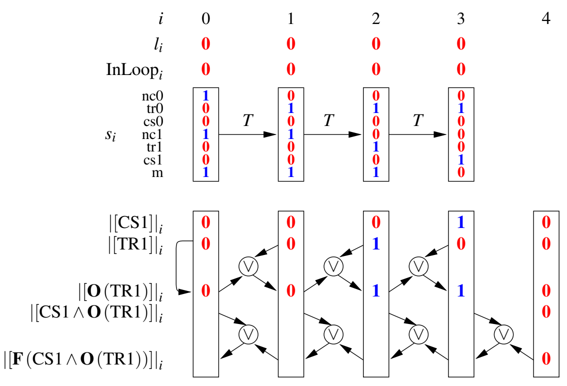
No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



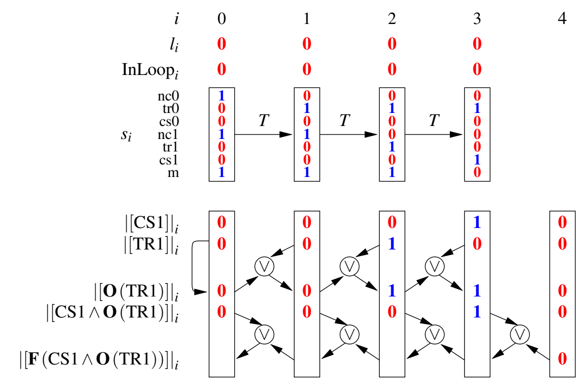
No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



No Virtual Unrolling Encoding: Illustration

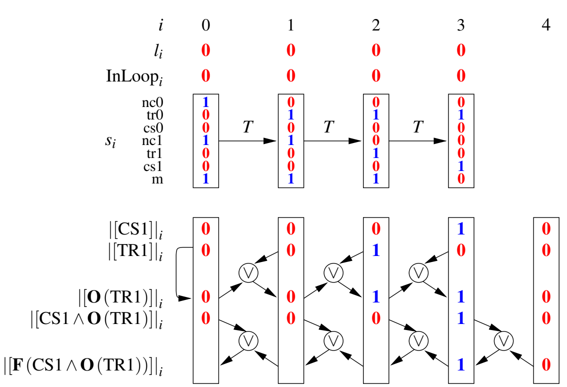
- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science Advanced Tutorial on Bounded Model Checking at ACSD'06 - ATPN'06, Keijo Heljanko and Tommi Junttila – 100/130

No Virtual Unrolling Encoding: Illustration

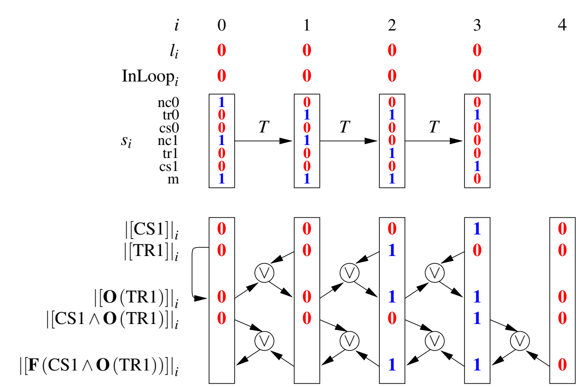
- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science Advanced Tutorial on Bounded Model Checking at ACSD'06 - ATPN'06, Keijo Heljanko and Tommi Junttila – 100/130

No Virtual Unrolling Encoding: Illustration

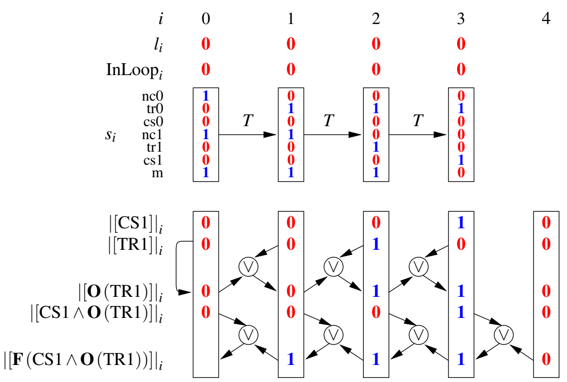
- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory for Theoretical Computer Science Advanced Tutorial on Bounded Model Checking at ACSD'06 - ATPN'06, Keijo Heljanko and Tommi Junttila – 100/130

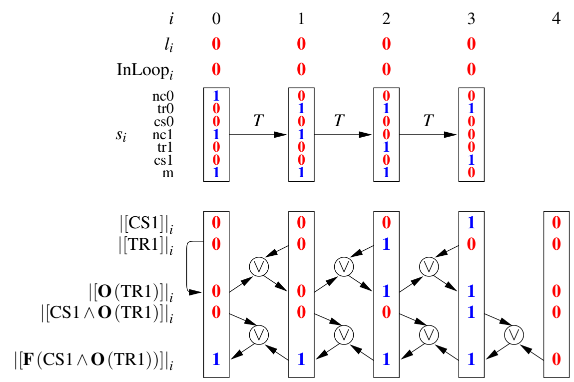
No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



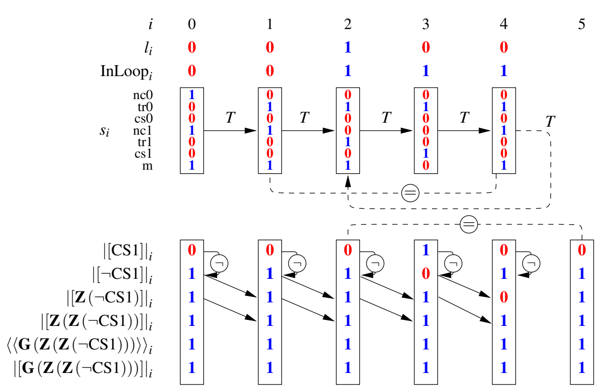
No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 3$, no loop
- Find witness for $\mathbf{F}(\mathrm{CS1} \wedge \mathbf{O}(\mathrm{TR1}))$



No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 4$, $l_2 = \mathbf{1}$
- Incorrectly evaluates $\mathbf{G}(\mathbf{Z}(\mathbf{Z}(\neg \mathrm{CS1})))$ to $\mathbf{1}$



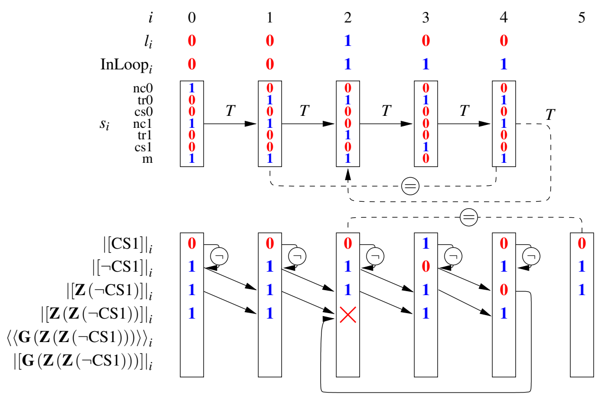
No Virtual Unrolling Encoding: Stabilization Test

- A fix for the problem: Stabilization Test
- Force that the first state in the loop sees two pasts: the previous state and the last state in the loop

φ	Added constraints for $1 \le i \le k$
$\mathbf{Y}\psi_1$	$l_i \Rightarrow (\lvert[\mathbf{Y}\psi_1]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_k)$
$\mathbf{Z}\psi_1$	$l_i \Rightarrow (\lvert[\mathbf{Z}\psi_1]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_k)$
$\mathbf{O}\psi_1$	$l_i \Rightarrow (\lvert[\mathbf{O}\psi_1]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_i \vee \lvert[\mathbf{O}\psi_1]\rvert_k)$
$\mathbf{H}\psi_1$	$l_i \Rightarrow (\lvert[\mathbf{H}\psi_1]\rvert_i \Leftrightarrow \lvert[\psi_1]\rvert_i \wedge \lvert[\mathbf{H}\psi_1]\rvert_k)$
$\psi_1 \mathbf{S} \psi_2$	$l_i \Rightarrow (\lvert[\psi_1 \mathbf{S} \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_2]\rvert_i \vee (\lvert[\psi_1]\rvert_i \wedge \lvert[\psi_1 \mathbf{S} \psi_2]\rvert_k))$
$\psi_1 \mathbf{T} \psi_2$	$l_i \Rightarrow (\lvert[\psi_1 \mathbf{T} \psi_2]\rvert_i \Leftrightarrow \lvert[\psi_2]\rvert_i \wedge (\lvert[\psi_1]\rvert_i \vee \lvert[\psi_1 \mathbf{T} \psi_2]\rvert_k))$

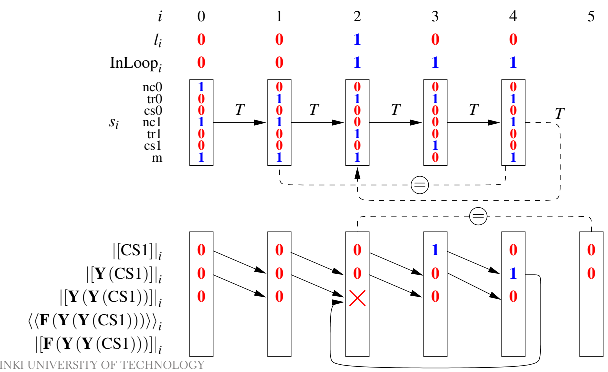
No Virtual Unrolling Encoding: Illustration

- Mutex example, $k = 4$, $l_2 = \mathbf{1}$
- Cannot evaluate $\mathbf{G}\,(\mathbf{Z}\,(\mathbf{Z}\,(\neg \mathrm{CS1})))$ to $\mathbf{1}$



No Virtual Unrolling Encoding: Illustration

- Cannot evaluate $\mathbf{F}\,(\mathbf{Y}\,(\mathbf{Y}\,(\mathrm{CS1})))$ to $\mathbf{1}$ although $\pi \models_k \mathbf{F}\,(\mathbf{Y}\,(\mathbf{Y}\,(\mathrm{CS1})))$
- That is, misses a minimal length counter-example



Encoding II: Virtual Unrolling

- Idea: virtually unroll the loop up to a "past operator depth" $\delta(\psi)$ of the formula ψ
- Translation is of size $O(|I| + k \cdot |T| + k \cdot |\psi| \cdot \delta(\psi))$
- Idea of virtual unrolling is from Benedetti&Cimatti TACAS'03 paper but our encoding is more compact
- Detects minimal-length counter-examples

Past Operator Depth

- The past operator depth of a PLTL formula is the maximum number of nested past operators in it

$$
\begin{array}{lcll}
\delta(\psi_1) & = & 0 & \text{for } \psi_1 \in AP \\
\delta(\circ\psi_1) & = & \delta(\psi_1) & \text{for } \circ \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\} \\
\delta(\psi_1 \circ \psi_2) & = & \max(\delta(\psi_1), \delta(\psi_2)) & \text{for } \circ \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}\} \\
\delta(\circ\psi_1) & = & 1 + \delta(\psi_1) & \text{for } \circ \in \{\mathbf{Y}, \mathbf{Z}, \mathbf{O}, \mathbf{H}\} \\
\delta(\psi_1 \circ \psi_2) & = & 1 + \max(\delta(\psi_1), \delta(\psi_2)) & \text{for } \circ \in \{\mathbf{S}, \mathbf{T}\}
\end{array}
$$

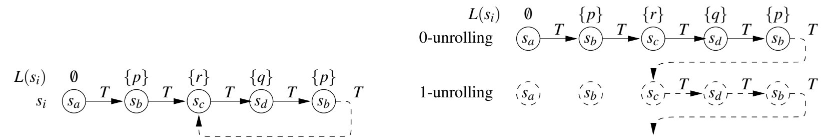
- E.g. $\delta(\mathbf{G}\,((\mathbf{Y}\,p)\,\mathbf{S}\,(\mathbf{H}\,q))) = 2$

Periods and d-Unrollings

- The period $p(\pi)$ of a (k,l)-loop path π is $k-l+1$, i.e. the number of states in the loop
- The jth, $j \geq 0$, state in π belongs to the d-unrolling of the loop if $d \geq 0$ is the smallest integer such that $j < l + ((d+1) \cdot p(\pi))$

Periods and d-Unrollings

- The period of the $(4,2)$-loop below is 3
- The 2nd state s_c belongs to the 0-unrolling and the 5th state s_c belongs to the 1-unrolling



(a) a $(4,2)$-loop (b) virtually unrolled

Stabilization Theorem

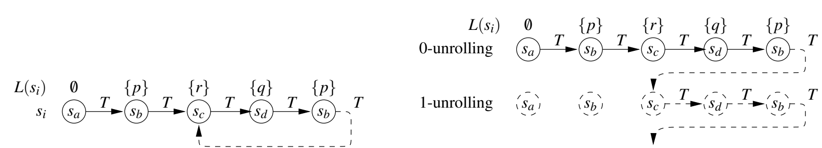
- Benedetti and Cimatti showed that a PLTL formula can only distinguish between corresponding time points in different unrollings up to the past operator depth of the formula
- Formally: if the time point i belongs to a d-unrolling of the loop with $d \geq \delta(\varphi)$ then:

$$\pi^i \models \varphi \text{ iff } \pi^j \models \varphi$$

where $j = i - ((d - \delta(\varphi)) \cdot p(\pi)$

Stabilization Theorem

- The $(4,2)$-loop $\pi = s_0 s_1 \ldots$ below
- $\pi^2 \not\models r \wedge \mathbf{O}\, q$ but $\pi^5 \models r \wedge \mathbf{O}\, q$, $\pi^8 \models r \wedge \mathbf{O}\, q$, and generally $\pi^{2+3 \cdot d} \models r \wedge \mathbf{O}\, q$ for all $d \geq 1$
- $\pi^2 \models \mathbf{H}\,(\neg q)$ but $\pi^5 \not\models \mathbf{H}\,(\neg q)$, $\pi^8 \not\models \mathbf{H}\,(\neg q)$, and generally $\pi^{2+3 \cdot d} \not\models \mathbf{H}\,(\neg q)$ for all $d \geq 1$



(a) a $(4,2)$-loop (b) virtually unrolled

Virtual Unrolling Encoding

- The generic form of our translation is as in the LTL case:

$$|[M]|_k \wedge |[\text{LoopConstraints}]|_k \wedge |[\text{LastStateConstraints}]|_k \wedge |[\psi, k]|_0$$

- As before, $|[M]|_k \equiv I(s_0) \wedge \bigwedge_{i=1}^{k} T(s_{i-1}, s_i)$
- Similarly, $|[\text{LoopConstraints}]|_k$ are as in the LTL case

Virtual Unrolling Encoding: Subformula Variables

- For each subformula φ of ψ and for each $0 \leq d \leq \delta(\varphi)$, introduce a variable $|[\varphi]|_i^d$ where $i \in \{0, 1, \ldots, k, k+1\}$
- $|[\varphi]|_i^d$ evaluates the value of the subformula φ at time index i in the d-unrolling
- Because of the stabilization theorem, we can define

$$|[\varphi]|_i^d \equiv |[\varphi]|_i^{\delta(\varphi)} \text{ for } d > \delta(\varphi)$$

Virtual Unrolling Encoding: Last State Constraints

- The no-loop case: force "pessimistic" future
- The (k,i)-loop case: connect the future state $k+1$ to the loop state i in the next unrolling

	$\vert[\text{LastStateConstraints}]\vert_k$ $0 \leq d \leq \delta(\phi)$
Base	$\neg\text{LoopExists} \Rightarrow (\vert[\phi]\vert_{k+1}^d \Leftrightarrow \mathbf{0})$
$1 \leq i \leq k$	$l_i \Rightarrow \left(\vert[\phi]\vert_{k+1}^d \Leftrightarrow \vert[\phi]\vert_i^{\min(d+1,\delta(\phi))}\right)$

Virtual Unrolling Encoding: Propositional Operators

- Encoding propositional operators is straighforward

φ	$0 \leq i \leq k, 0 \leq d \leq \delta(\varphi)$
p	$\vert[p]\vert_i^d \Leftrightarrow p_i$
$\neg p$	$\vert[\neg p]\vert_i^d \Leftrightarrow \neg p_i$
$\psi_1 \wedge \psi_2$	$\vert[\psi_1 \wedge \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_1]\vert_i^d \wedge \vert[\psi_2]\vert_i^d$
$\psi_1 \vee \psi_2$	$\vert[\psi_1 \vee \psi_2]\vert_i^d \Leftrightarrow \vert[\psi_1]\vert_i^d \vee \vert[\psi_2]\vert_i^d$

Virtual Unrolling Encoding: Past Operators

- 0-unrolling case similar to the “No unrolling” encoding

	$i=0, d=0$	$1 \le i \le k, d=0$
$\|[\mathbf{Y}\,\psi_1]\|_i^d$	$\mathbf{0}$	$\|[\psi_1]\|_{i-1}^d$
$\|[\mathbf{Z}\,\psi_1]\|_i^d$	$\mathbf{1}$	$\|[\psi_1]\|_{i-1}^d$
$\|[\mathbf{O}\,\psi_1]\|_i^d$	$\|[\psi_1]\|_i^d$	$\|[\psi_1]\|_i^d \vee \|[\mathbf{O}\,\psi_1]\|_{i-1}^d$
$\|[\mathbf{H}\,\psi_1]\|_i^d$	$\|[\psi_1]\|_i^d$	$\|[\psi_1]\|_i^d \wedge \|[\mathbf{H}\,\psi_1]\|_{i-1}^d$
$\|[\psi_1\,\mathbf{S}\,\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \vee \left(\|[\psi_1]\|_i^d \wedge \|[\psi_1\,\mathbf{S}\,\psi_2]\|_{i-1}^d\right)$
$\|[\psi_1\,\mathbf{T}\,\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \wedge \left(\|[\psi_1]\|_i^d \vee \|[\psi_1\,\mathbf{T}\,\psi_2]\|_{i-1}^d\right)$

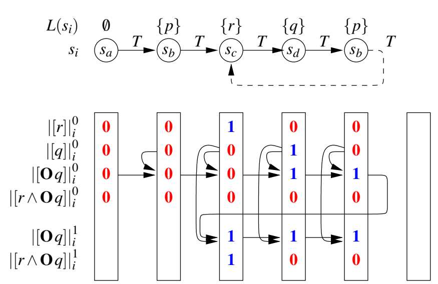
Virtual Unrolling Encoding: Past Operators

- d-unrolling, $d \ge 1$, similar except that the loop point looks back to the last state in the previous unrolling
- We use $ite(c,t,e)$ for $(c \Rightarrow t) \wedge (\neg c \Rightarrow e)$

ϕ	$i=0, 1 \le d \le \delta(\phi)$	$1 \le i \le k, 1 \le d \le \delta(\phi)$
$\|[\mathbf{Y}\,\psi_1]\|_i^d$	$\mathbf{0}$	$ite(l_i, \|[\psi_1]\|_k^{d-1}, \|[\psi_1]\|_{i-1}^d)$
$\|[\mathbf{Z}\,\psi_1]\|_i^d$	$\mathbf{1}$	$ite(l_i, \|[\psi_1]\|_k^{d-1}, \|[\psi_1]\|_{i-1}^d)$
$\|[\mathbf{O}\,\psi_1]\|_i^d$	$\|[\psi_1]\|_i^d$	$\|[\psi_1]\|_i^d \vee ite(l_i, \|[\mathbf{O}\,\psi_1]\|_k^{d-1}, \|[\mathbf{O}\,\psi_1]\|_{i-1}^d)$
$\|[\mathbf{H}\,\psi_1]\|_i^d$	$\|[\psi_1]\|_i^d$	$\|[\psi_1]\|_i^d \wedge ite(l_i, \|[\mathbf{H}\,\psi_1]\|_k^{d-1}, \|[\mathbf{H}\,\psi_1]\|_{i-1}^d)$
$\|[\psi_1\,\mathbf{S}\,\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \vee \left(\|[\psi_1]\|_i^d \wedge ite(l_i, \|[\psi_1\,\mathbf{S}\,\psi_2]\|_k^{d-1}, \|[\psi_1\,\mathbf{S}\,\psi_2]\|_{i-1}^d)\right)$
$\|[\psi_1\,\mathbf{T}\,\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d$	$\|[\psi_2]\|_i^d \wedge \left(\|[\psi_1]\|_i^d \vee ite(l_i, \|[\psi_1\,\mathbf{T}\,\psi_2]\|_k^{d-1}, \|[\psi_1\,\mathbf{T}\,\psi_2]\|_{i-1}^d)\right)$

Virtual Unrolling Encoding: Illustration

- Evaluating $r \wedge \mathbf{O}\,q$ with virtual unrolling



Virtual Unrolling Encoding: Future Operators

- Similar to the pure LTL encoding

φ	$0 \le i \le k, 0 \le d \le \delta(\varphi)$
$\mathbf{X}\,\phi$	$\|[\mathbf{X}\,\phi]\|_i^d \Leftrightarrow \|[\phi]\|_{i+1}^d$
$\mathbf{F}\,\phi$	$\|[\mathbf{F}\,\phi]\|_i^d \Leftrightarrow \|[\phi]\|_i^d \vee \|[\mathbf{F}\,\phi]\|_{i+1}^d$
$\mathbf{G}\,\phi$	$\|[\mathbf{G}\,\phi]\|_i^d \Leftrightarrow \|[\phi]\|_i^d \wedge \|[\mathbf{G}\,\phi]\|_{i+1}^d$
$\psi_1\,\mathbf{U}\,\psi_2$	$\|[\psi_1\,\mathbf{U}\,\psi_2]\|_i^d \Leftrightarrow \|[\psi_2]\|_i^d \vee \left(\|[\psi_1]\|_i^d \wedge \|[\psi_1\,\mathbf{U}\,\psi_2]\|_{i+1}^d\right)$
$\psi_1\,\mathbf{R}\,\psi_2$	$\|[\psi_1\,\mathbf{R}\,\psi_2]\|_i^d \Leftrightarrow \|[\psi_2]\|_i^d \wedge \left(\|[\psi_1]\|_i^d \vee \|[\psi_1\,\mathbf{R}\,\psi_2]\|_{i+1}^d\right)$

Virtual Unrolling Encoding: Auxiliary Encoding

- The (k,l)-loop cases require an auxiliary encoding to force the cyclic dependencies to evaluate correctly
- Similarly to the pure LTL case except that $\langle\langle \mathbf{F}\,\phi\rangle\rangle_k$ and $\langle\langle \mathbf{G}\,\phi\rangle\rangle_k$ are evaluated by using the stabilized values in the last unrolling

Base	$\langle\langle \mathbf{F}\,\phi\rangle\rangle_0 \Leftrightarrow \mathbf{0}$ $\langle\langle \mathbf{G}\,\phi\rangle\rangle_0 \Leftrightarrow \mathbf{1}$
$1 \le i \le k$	$\langle\langle \mathbf{F}\,\phi\rangle\rangle_i \Leftrightarrow \langle\langle \mathbf{F}\,\phi\rangle\rangle_{i-1} \vee \left(\mathsf{InLoop}_i \wedge |[\phi]|_i^{\delta(\phi)}\right)$ $\langle\langle \mathbf{G}\,\phi\rangle\rangle_i \Leftrightarrow \langle\langle \mathbf{G}\,\phi\rangle\rangle_{i-1} \wedge \neg\left(\mathsf{InLoop}_i \wedge \neg|[\phi]|_i^{\delta(\phi)}\right)$

Virtual Unrolling Encoding: Auxiliary Encoding

- Force cyclic dependencies to evaluate correctly

ϕ	Added constraint
$\mathbf{F}\,\psi_1$	$\mathsf{LoopExists} \Rightarrow \left(|[\mathbf{F}\,\psi_1]|_k^{\delta(\phi)} \Rightarrow \langle\langle \mathbf{F}\,\psi_1\rangle\rangle_k\right)$
$\mathbf{G}\,\psi_1$	$\mathsf{LoopExists} \Rightarrow \left(|[\mathbf{G}\,\psi_1]|_k^{\delta(\phi)} \Leftarrow \langle\langle \mathbf{G}\,\psi_1\rangle\rangle_k\right)$
$\psi_1\,\mathbf{U}\,\psi_2$	$\mathsf{LoopExists} \Rightarrow \left(|[\psi_1\,\mathbf{U}\,\psi_2]|_k^{\delta(\phi)} \Rightarrow \langle\langle \mathbf{F}\,\psi_2\rangle\rangle_k\right)$
$\psi_1\,\mathbf{R}\,\psi_2$	$\mathsf{LoopExists} \Rightarrow \left(|[\psi_1\,\mathbf{R}\,\psi_2]|_k^{\delta(\phi)} \Leftarrow \langle\langle \mathbf{G}\,\psi_2\rangle\rangle_k\right)$

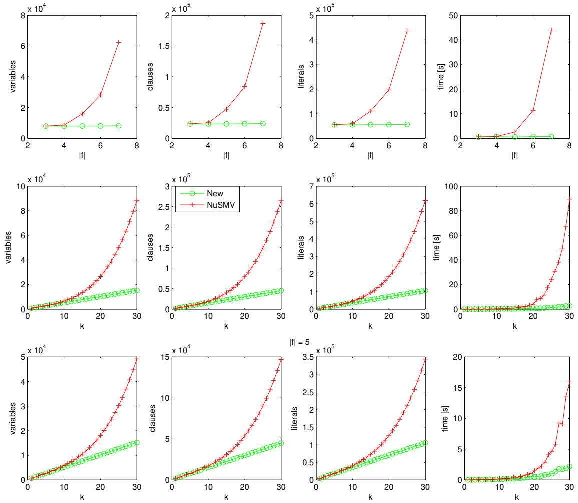
Virtual Unrolling Encoding: Some Experiments

- From our VMCAI'05 paper
- Similar (but not exactly same) encoding as here
- 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.

Virtual Unrolling Encoding: Benchmarks

M	k	NuSMV				New			
		vars	*clauses*	*time*	Σ*time*	*vars*	*clauses*	*time*	Σ*time*
VMCAI2005/abp	16	25,175	74,208	104	342	22,827	67,116	52.5	269
VMCAI2005/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
VMCAI2005/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
VMCAI2005/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
VMCAI2005/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

Virtual Unrolling Encoding: Benchmarks II



Virtual Unrolling Encoding: Some Concluding Remarks

- For formulas with no past operators, the encoding is equivalent to the pure LTL encoding
- Sometimes produces shorter counter-examples than the “no unrolling” encoding
  This may be beneficial for models with a complex transition relation
- Sometimes faster, sometimes slower than the “no unrolling” encoding, no clear winner
- Easy to make incremental and complete by extending the previously presented techniques

Some Additional References

BMC beoynd LTL

- Heljanko, K., Junttila, T., Keinänen, M., Lange, M., and Latvala, T.: Bounded Model Checking for Weak Alternating Büchi Automata. CAV’06
  - A BMC procedure for all ω-regular languages by using WABAs, enables BMC for a subset of PSL extending LTL

BMC for Branching Time Temporal Logics

- Penczek, W., Wozna, B., and Zbrzezny, A.: Bounded Model Checking for the Universal Fragment of CTL. Fundamenta Informatica 51(1–2):135–156, 2002.
  - A BMC procedure for the universal fragment of a branching time temporal logic
- Wozna, B.: ACTL* properties abd Bounded Model Checking. Fundamenta Informatica 63(1):65–87, 2004.
  - A BMC procedure for the universal fragment of a branching time temporal logic subsuming ACTL and LTL

BMC by using Extensions of Propositional SAT

- SMT-LIB: The Satisfiability Modulo Theories Library. `http://combination.cs.uiowa.edu/smtlib/`
  - Benchmarks, links to solvers etc. for the SAT modulo theories problem
- Audemard, G., Cimatti, A., Kornilowicz, A., and Sebastiani, R.: Bounded Model Checking for Timed Systems. FORTE'02.
  - BMC for timed automata (direct LTL encoding)

BMC by using Extensions of Propositional SAT

- Sorea, M.: Bounded Model Checking for Timed Automata. Electronic Notes in Theor. Comp. Sc. 68(5),2005.
  - BMC for timed automata (Büchi automata based LTL encoding)
- Audemard, G., Bozzano, M., Cimatti, A., and Sebastiani, R.: Verifying Industrial Hybrid Systems with MathSAT. Electronic Notes in Theor. Comp. Sc. 119:17–32,2005.
  - BMC for linear hybrid automata

Conclusions of Tutorial Part 2

- We have presented
  - a simple and compact BMC encoding for LTL
  - two simple BMC encodings for PLTL:
    1. one that is very compact but does not always detect minimal-length counterexamples, and
    2. one that is less compact but detects minimal-length counter-examples
  - how to exploit incremental SAT solvers in BMC
  - an approach to make BMC for (P)LTL complete