Bounded Model Checking for Finite-State Systems

Copenhagen, 2 March 2010

Quantitative Model Checking PhD School

Keijo Heljanko
Aalto University
Keijo.Heljanko@tkk.fi
Co-Author of Slides

Many of the slides used in this tutorial are from Advanced Tutorial on Bounded Model Checking at ACSD 2006 / Petri Nets 2006, co-authored with my colleague:

- D.Sc. (Tech.) Tommi Junttila
  - Email: Tommi.Junttila@tkk.fi
  - Homepage: http://users.ics.tkk.fi/tjunttil

Many thanks to Tommi for letting me use also his slides in preparing this tutorial.
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 $M = (S, s_{\text{init}}, T, L)$, where
  - $S$ is a finite set of states,
  - $s_{\text{init}} \in S$ 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
  - $L : S \rightarrow 2^{AP}$ is a valuation, i.e. a function which maps each state to those atomic propositions which hold in that state.
Running Example: Mutex

- $AP = \{NC0, TR0, CS0, NC1, TR1, CS1\}$
- The Kripke structure of our running example is:

![Kripke structure diagram]

$L = \{NC0, NC1\}$

$L = \{TR0, NC1\}$

$L = \{CS0, NC1\}$

$L = \{CS0, TR1\}$

$L = \{TR0, TR1\}$

$L = \{TR0, CS1\}$

$L = \{NC0, CS1\}$

$L = \{NC0, TR1\}$

$L = \{NC0, NC1\}$
A path in a Kripke structure $M = (S, s_{init}, T, L)$ is an infinite sequence $\pi = s_0s_1 \ldots$ 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 $\pi = s_0s_1 \ldots$ is a $(k, l)$-loop if
$\pi = (s_0s_1 \ldots s_{l-1})(s_l \ldots s_k)^\omega$ such that $0 < l \leq k$ and $s_{l-1} = s_k$

If $\pi$ is a $(k, l)$-loop, then it is a $(k + 1, l + 1)$-loop
The dashed path in the figure is a \((4, 2)\)-loop as it equals to

\[
\{nc0, nc1, m\} \{tr0, nc1, m\} (\{tr0, tr1, m\} \{tr0, cs1\} \{tr0, nc1, m\})^\omega
\]
LTL Syntax

- Each \( p \in AP \) is an LTL formula

- If \( \psi_1 \) and \( \psi_2 \) are LTL formulae, then the following are LTL formulae:
  - \( \neg \psi_1 \)  
    - negation
  - \( \psi_1 \lor \psi_2 \)  
    - disjunction
  - \( \psi_1 \land \psi_2 \)  
    - conjunction
  - \( X \psi_1 \)  
    - “next”
  - \( F \psi_1 \)  
    - “finally” (or “eventually”)
  - \( G \psi_1 \)  
    - “globally” (or “always”)
  - \( \psi_1 U \psi_2 \)  
    - “until”
  - \( \psi_1 R \psi_2 \)  
    - “release”
Examples of LTL formulae

- Invariance:
  \[ G \neg (CS0 \land CS1) \]

- Process 0 always finally leaves the critical section:
  \[ G (CS0 \Rightarrow F (\neg CS0)) \]

- “Justice” fairness (infinitely often):
  \[ GF (CS0) \]

- “Weak” fairness:
  \[ (FG (TR0)) \Rightarrow (GF (CS0)) \]

- “Strong” fairness:
  \[ (GF (TR0)) \Rightarrow (GF (CS0)) \]
Semantics of LTL

- Let $\pi = s_0s_1 \ldots$ be a path with labelling $L(s_i) \in 2^{AP}$
- The relation $\pi^i \models \psi$ for “$\psi$ holds at time point $i$ in $\pi$”:

  $\pi^i \models \psi \iff \psi \in L(s_i)$ for $\psi \in AP$
  $\pi^i \models \neg \psi \iff \pi^i \not\models \psi$
  $\pi^i \models \psi_1 \lor \psi_2 \iff \pi^i \models \psi_1$ or $\pi^i \models \psi_2$
  $\pi^i \models \psi_1 \land \psi_2 \iff \pi^i \models \psi_1$ and $\pi^i \models \psi_2$
  $\pi^i \models X\psi \iff \pi^{i+1} \models \psi$
  $\pi^i \models F\psi_1 \iff \exists n \geq i : \pi^n \models \psi_1$
  $\pi^i \models G\psi_1 \iff \forall n \geq i : \pi^n \models \psi_1$
  $\pi^i \models \psi_1 U \psi_2 \iff \exists n \geq i : (\pi^n \models \psi_2 \land \forall i \leq j < n : \pi^j \models \psi_1)$
  $\pi^i \models \psi_1 R \psi_2 \iff (\forall n \geq i : \pi^n \models \psi_2) \lor (\exists n \geq i : \pi^n \models \psi_1 \land \forall i \leq j \leq n : \pi^j \models \psi_2)$
Semantics of LTL

\[ L(s_i) \{P\} \quad \{P\} \quad \{Q\} \quad \{P,Q\} \quad \{Q\} \]

- \( \pi_0 \models P, \pi_0 \models Q, \pi_2 \models Q \)
- \( \pi_0 \models P \mathsf{U} Q, \pi_0 \models Q \mathsf{R} P \)
- \( \pi_0 \models F Q, \pi_0 \models G P \)
- \( \pi_2 \models G Q \)
- \( \pi_0 \models F G Q \)
- \( \pi_0 \models G F P \)
Semantics of LTL

- We write \( \pi \models \psi \) if \( \pi^0 \models \psi \) and say that \( \pi \) is a witness path for \( \psi \).

- An LTL formula \( \psi \) holds in a Kripke structure \( M = (S, s_{init}, T, L) \) if \( \pi \models \psi \) for each path \( \pi \) in \( M \).

- Model checking problem: find whether \( M \models \psi \).

- Dually: is there a counter-example path \( \pi \) in \( M \) such that \( \pi \models \neg \psi \)?
  - If there is, then \( M \not\models \psi \).
  - Otherwise, \( M \models \psi \).
The dashed path below is a witness for $G(\neg CS0)$ and thus a counter-example for $\neg G(\neg CS0) \equiv F(CS0)$.
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
No-Loop Case: Safety Properties

- No-loop case is tailored to detect counterexamples to safety properties with small bounds.
- Consider the no-loop case above.
- We know that $\pi \models F q$ for each infinite extension $\pi$.
- But we don’t know whether $\pi \models G p$ for any infinite extension $\pi$.
- To formalize this, we need bounded semantics of 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:

\[ \neg(\psi_1 \lor \psi_2) \equiv (\neg \psi_1) \land (\neg \psi_2) \]
\[ \neg(\psi_1 \land \psi_2) \equiv (\neg \psi_1) \lor (\neg \psi_2) \]
\[ \neg(\neg \psi) \equiv \psi \]
\[ \neg(X\psi) \equiv X(\neg \psi) \]
\[ \neg(\psi_1 U \psi_2) \equiv (\neg \psi_1) R (\neg \psi_2) \]
\[ \neg(\psi_1 R \psi_2) \equiv (\neg \psi_1) U (\neg \psi_2) \]
Bounded Semantics of LTL

Given a path $\pi = s_0s_1 \ldots$ and a bound $k \geq 0$, $\pi \models_k \psi$ iff (i) $\pi$ is a $(k, l)$-loop and $\pi^0 \models \psi$, or (ii) $\pi^0 \models_{nl} \psi$, where:

- $\pi^i \models_{nl} p \iff p \in L(s_i)$ for $p \in AP$
- $\pi^i \models_{nl} \neg p \iff p \notin L(s_i)$ for $p \in AP$
- $\pi^i \models_{nl} \psi_1 \lor \psi_2 \iff \pi^i \models_{nl} \psi_1$ or $\pi^i \models_{nl} \psi_2$
- $\pi^i \models_{nl} \psi_1 \land \psi_2 \iff \pi^i \models_{nl} \psi_2$ and $\pi^i \models_{nl} \psi_2$
- $\pi^i \models_{nl} X\psi_1 \iff i < k$ and $\pi^{i+1} \models_{nl} \psi_1$
- $\pi^i \models_{nl} F\psi_1 \iff \exists i \leq n \leq k : \pi^n \models_{nl} \psi_1$
- $\pi^i \models_{nl} G\psi_1 \iff \bot$
- $\pi^i \models_{nl} \psi_1 U \psi_2 \iff \exists i \leq n \leq k : (\pi^n \models_{nl} \psi_2 \land \forall i \leq j < n : \pi^j \models_{nl} \psi_1)$
- $\pi^i \models_{nl} \psi_1 R \psi_2 \iff \exists i \leq n \leq k : (\pi^n \models_{nl} \psi_1 \land \forall i \leq j \leq n : \pi^j \models_{nl} \psi_2)$
Bounded Semantics of LTL

- $\models_k$ under-approximates $\models$.

- If $\pi \models_k \psi$, then $\pi \models \psi$.

- For each ultimately periodic path $\pi$ there is a $k$ such that $\pi$ 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 Encoding for LTL

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

- The generic form of $\left|M, \psi, k\right|$ is

$$\left|M\right|_k \land \left|\psi, k\right|_0$$

- As before, $\left|M\right|_k \equiv I(s_0) \land \bigwedge_{i=1}^{k} T(s_{i-1}, s_i)$ encodes paths by unrolling transition relation $k$ times

- $\left|\psi, k\right|_0$ constraints paths to be witnesses for $\psi$ under the bounded semantics
Our Approach: Simple BMC

  - Incremental and complete version of the encoding for LTL with past time operators

  - Survey of linear LTL encodings for BMC, including also approaches based on Büchi automata based LTL model checking
BMC for LTL: Some Related Work


- Cimatti, A., Pistore, M., Roveri, M., and Sebastiani, R.: Improving 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: $\left[ M \right]_k \land \left[ \psi, k \right]$

(a) no loop

(b) $(k, l)$-loop

- Basic idea: $\left[ \psi, k \right] \equiv -\left[ \psi, k \right]_0 \lor \bigvee_{l=1}^{k} l \left[ \psi, k \right]_0$

where

- $-\left[ \psi, k \right]_0$ evaluates $\psi$ in the no loop case
- $l \left[ \psi, k \right]_0$ evaluates $\psi$ 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 $\pi$ such that $\pi \models_k \psi$

- The generic form of our translation is

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

- As before, $| [M] |_k \equiv I(s_0) \land \bigwedge_{i=1}^{k} T(s_{i-1}, s_i)$

- Seen as a Boolean circuit, $| [M, \psi, k] |$ is of size $O(|I| + k \cdot |T| + k \cdot |\psi|)$
Loop Constraints

(a) no loop

- 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

(b) \((k, l)\)-loop
Loop Constraints

<table>
<thead>
<tr>
<th></th>
<th>$\lbrack \text{LoopConstraints} \rbrack_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base</strong></td>
<td></td>
</tr>
<tr>
<td>$l_0$</td>
<td>$\iff \bot$</td>
</tr>
<tr>
<td>$\text{InLoop}_0$</td>
<td>$\iff \bot$</td>
</tr>
<tr>
<td><strong>1 \leq i \leq k</strong></td>
<td></td>
</tr>
<tr>
<td>$l_i$</td>
<td>$\Rightarrow (s_{i-1} = s_k)$</td>
</tr>
<tr>
<td>$\text{InLoop}_i$</td>
<td>$\iff \text{InLoop}_{i-1} \lor l_i$</td>
</tr>
<tr>
<td>$l_i$</td>
<td>$\Rightarrow \neg \text{InLoop}_{i-1}$</td>
</tr>
<tr>
<td>LoopExists</td>
<td>$\iff \text{InLoop}_k$</td>
</tr>
</tbody>
</table>

- $\text{InLoop}_i$ 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
  \( \{nc0, nc1, m\} \{tr0, nc1, m\} \{tr0, tr1, m\} \{tr0, cs1\} \)

\[
\begin{array}{c|cccc}
  i & 0 & 1 & 2 & 3 & 4 \\
  l_i & \bot & \bot & \bot & \bot & \bot \\
  InLoop_i & \bot & \bot & \bot & \bot & \bot \\
  s_i & \begin{array}{c}
  nc0 \\
  tr0 \\
  cs0 \\
  nc1 \\
  tr1 \\
  cs1 \\
  m
  \end{array} & \begin{array}{c}
  \bot
  \end{array} & \begin{array}{c}
  \bot
  \end{array} & \begin{array}{c}
  \bot
  \end{array} & \begin{array}{c}
  \bot
  \end{array}
\end{array}
\]

\[\begin{array}{c}
T \quad T \quad T
\end{array}\]
Illustration of the Encoding

- Mutex example, $k = 4$, $l_2 = \top$
- The $(4, 2)$-loop

$$\{nc0, nc1, m\} \{tr0, nc1, m\} (\{tr0, tr1, m\} \{tr0, cs1\} \{tr0, nc1, m\})^\omega$$
For each subformula $\phi$ of $\psi$, introduce a variable $\lbrack\phi\rbrack_i$, where $i \in \{0, 1, \ldots, k, k + 1\}$

$\lbrack\phi\rbrack_i$ evaluates the value of the subformula $\phi$ at time step $i$

Thus $\lbrack\psi\rbrack_0$ 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 $k$th index
Encoding LTL: Last State Constraints

- The no-loop case: force “pessimistic” future: all formulas evaluate to $\bot$

- The $(k, i)$-loop case: connect the future state $k + 1$ to the loop state $i$

<table>
<thead>
<tr>
<th>$1 \leq i \leq k$</th>
<th>$l_i \Rightarrow ([\phi]_{k+1} \iff [\phi]_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>$\neg \text{LoopExists} \Rightarrow ([\phi]_{k+1} \iff \bot)$</td>
</tr>
</tbody>
</table>

$$[[\text{LastStateConstraints}]_k$$
Encoding propositional operators is straightforward

<table>
<thead>
<tr>
<th>0 ≤ i ≤ k</th>
<th>φ</th>
<th>constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>$p_i$</td>
<td>$[[p]]_i$ ⇔ $p_i$</td>
</tr>
<tr>
<td>$\neg p$</td>
<td>$\neg p_i$</td>
<td>$[[\neg p]]_i$ ⇔ $\neg p_i$</td>
</tr>
<tr>
<td>$\psi_1 \land \psi_2$</td>
<td>$[[\psi_1 \land \psi_2]]_i$</td>
<td>$[[\psi_1 \land \psi_2]]_i$ ⇔ $[[\psi_1]]_i \land [[\psi_2]]_i$</td>
</tr>
<tr>
<td>$\psi_1 \lor \psi_2$</td>
<td>$[[\psi_1 \lor \psi_2]]_i$</td>
<td>$[[\psi_1 \lor \psi_2]]_i$ ⇔ $[[\psi_1]]_i \lor [[\psi_2]]_i$</td>
</tr>
</tbody>
</table>
## 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

<table>
<thead>
<tr>
<th>(\phi)</th>
<th>encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X\phi)</td>
<td>(\llbracket X\phi \rrbracket_i \Leftrightarrow \llbracket \phi \rrbracket_{i+1})</td>
</tr>
<tr>
<td>(F\phi)</td>
<td>(\llbracket F\phi \rrbracket_i \Leftrightarrow \llbracket \phi \rrbracket_i \lor \llbracket F\phi \rrbracket_{i+1})</td>
</tr>
<tr>
<td>(G\phi)</td>
<td>(\llbracket G\phi \rrbracket_i \Leftrightarrow \llbracket \phi \rrbracket_i \land \llbracket G\phi \rrbracket_{i+1})</td>
</tr>
<tr>
<td>(\psi_1 U \psi_2)</td>
<td>(\llbracket \psi_1 U \psi_2 \rrbracket_i \Leftrightarrow \llbracket \psi_2 \rrbracket_i \lor (\llbracket \psi_1 \rrbracket_i \land \llbracket \psi_1 U \psi_2 \rrbracket_{i+1}))</td>
</tr>
<tr>
<td>(\psi_1 R \psi_2)</td>
<td>(\llbracket \psi_1 R \psi_2 \rrbracket_i \Leftrightarrow \llbracket \psi_2 \rrbracket_i \land (\llbracket \psi_1 \rrbracket_i \lor \llbracket \psi_1 R \psi_2 \rrbracket_{i+1}))</td>
</tr>
</tbody>
</table>
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 \(\phi\) evaluates to true at least once in the selected loop.
- Idea: \(\langle\langle \mathbf{G} \phi \rangle\rangle_k\) evaluates to true iff \(\phi\) evaluates to true in all states in the selected loop.

<table>
<thead>
<tr>
<th>Base</th>
<th>(\langle\langle \mathbf{F} \phi \rangle\rangle_0 \iff \bot)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\langle\langle \mathbf{G} \phi \rangle\rangle_0 \iff \top)</td>
</tr>
<tr>
<td>(1 \leq i \leq k)</td>
<td>(\langle\langle \mathbf{F} \phi \rangle\rangle_i \iff \langle\langle \mathbf{F} \phi \rangle\rangle_{i-1} \lor (\text{InLoop}_i \land [\phi]_i))</td>
</tr>
<tr>
<td></td>
<td>(\langle\langle \mathbf{G} \phi \rangle\rangle_i \iff \langle\langle \mathbf{G} \phi \rangle\rangle_{i-1} \land \neg(\text{InLoop}_i \land \neg[\phi]_i))</td>
</tr>
</tbody>
</table>
Force cyclic dependencies to evaluate correctly

<table>
<thead>
<tr>
<th></th>
<th>Added constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi$</td>
<td></td>
</tr>
<tr>
<td>$F\psi_1$</td>
<td>LoopExists $\Rightarrow (</td>
</tr>
<tr>
<td>$G\psi_1$</td>
<td>LoopExists $\Rightarrow (</td>
</tr>
<tr>
<td>$\psi_1 U \psi_2$</td>
<td>LoopExists $\Rightarrow (</td>
</tr>
<tr>
<td>$\psi_1 R \psi_2$</td>
<td>LoopExists $\Rightarrow (</td>
</tr>
</tbody>
</table>

Similar to using Büchi automata acceptance sets for ensuring the correct semantics of until formulas on infinite words.
BMC and Incremental SAT Solving

- SAT problems from BMC with increasing bounds are quite similar:
  \[ |[M, \psi, 0]| \preceq |[M, \psi, 1]| \preceq |[M, \psi, 2]| \preceq \ldots \]

- 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
Basic Approach to Incrementality

- Divide the BMC encoding into three parts:
  - Base encoding $\alpha$ - stays the same for all bounds
  - $k$-invariant part $\beta_i$ - is independent of the actual value of the bound $k$
  - $k$-dependent part $\gamma_i$ - is dependent on the value of the bound $k$

- Example of increasing bound from 3 to 4:
  - $\alpha \land \beta_0 \land \beta_1 \land \beta_2 \land \gamma_2$
  - $\alpha \land \beta_0 \land \beta_1 \land \beta_2 \land \beta_3 \land \gamma_3$
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: minimize the size of the $k$-dependent part.
Incrementality: Experimental Results

- From our CAV’05 paper. Approach integrated into NuSMV 2.4 as the “sbmc” algorithm
- 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

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

- **Best**
- **Worst**
Experiments, part 2

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

- Eén, N. and Sörensson N.: Temporal Induction by Incremental SAT Solving. \textit{BMC’03}.
  - An incremental and complete BMC procedure for invariants.

  - Incremental version of Benedetti-Cimatti translation for PLTL
BMC beyond LTL

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

  - A BMC procedure to solve bounded problems on context free grammars
BMC for Branching Time

- A BMC procedure for the universal fragment of a branching time temporal logic subsuming ACTL and LTL
BMC by using Extensions of Propositional SAT

  - Benchmarks, links to solvers etc. for the SAT modulo theories problem

  - BMC for timed automata (direct LTL encoding)
BMC by using Extensions of Propositional SAT

  - BMC for timed automata

  - BMC for linear hybrid automata

  - A bounded model checker HySAT for hybrid systems
Multicore BMC Engine: Tarmo

- Multicore BMC is an active research topic
- Utilizes randomization, sharing learned clauses between SAT solver instances solving a sequence of incremental BMC instances, and solver portfolio techniques to diversify search
- Works with any incremental BMC encoding
Tarmo: Multicore BMC Experiments

![Graph showing performance comparison between CONV and 4xCONV instances solved over time.](image)
Tarmo: Multicore BMC Experiments

![Graph showing instances solved over time for different methods: CONV, 4xCONV, and MULTICONV-SIMPLE.](Image)

- **CONV**
- **4xCONV**
- **MULTICONV-SIMPLE**

The graph illustrates the time (in seconds) taken to solve instances as the number of instances solved increases.
Tarmo: Multicore BMC Experiments

![Graph showing the comparison of different multicore BMC experiments. The x-axis represents the number of instances solved, ranging from 50 to 100, and the y-axis represents the time in seconds, ranging from 0 to 3600. The graph compares CONV, 4xCONV, MULTICONV-SIMPLE, and MULTICONV-FULL methods.]
Tarmo: Multicore BMC Experiments

![Graph showing the time taken to solve instances for MULTICONV-FULL](image)

- Time (s)
- Instances solved
Tarmo: Multicore BMC Experiments

![Graph showing time (s) vs. instances solved for MULTICONV-FULL and MULTICONV-ADAPTIVE]
Tarmo: Multicore BMC Experiments

![Graph showing the comparison of different BMC methods.](image-url)

- **MULTICONV-FULL**
- **MULTICONV-ADAPTIVE**
- **MULTICONV-TARMO**
- **MULTIBOUND-TARMO**

The graph compares the time (s) taken to solve instances as a function of the number of instances solved.
Conclusions of BMC Tutorial

- Bounded model checking is an alternative method for model checking of finite state systems
- The approach is best at “bug hunting” but can also be made complete
- In asynchronous systems different encodings of the transition relation have large performance differences
- Capturing LTL safety counterexamples is very useful in BMC
- Incremental SAT solving gives BMC a nice performance boost