Bounded Model Checking for Finite-State Systems

Copenhagen, 2 March 2010

Quantitative Model Checking PhD School

Keijo Heljanko

Aalto University

Keijo.Heljanko@tkk.fi

Aalto University

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.

Aalto University

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.

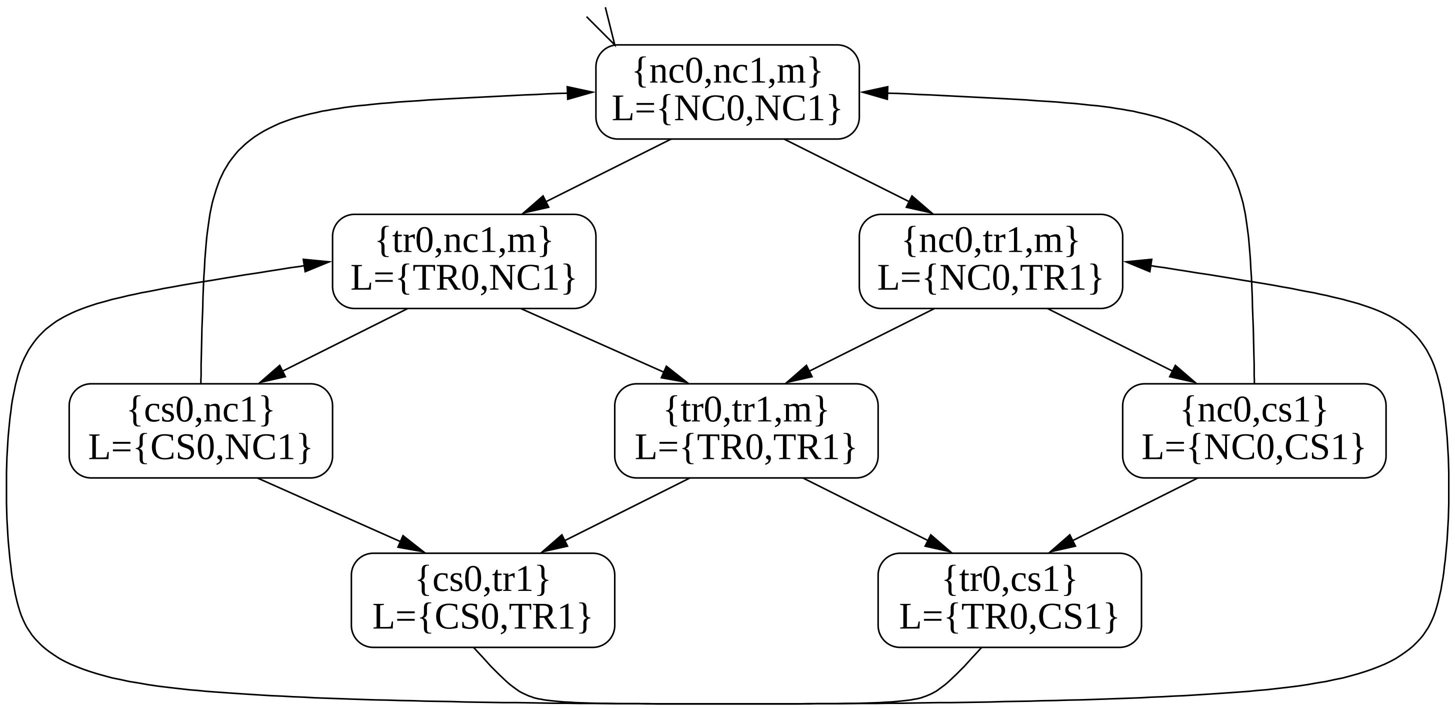
Formal Definition

- 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
  - $L : S \rightarrow 2^{AP}$ is a valuation, i.e. a function which maps each state to those atomic propositions which hold in that state.

Aalto University

Running Example: Mutex

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



Aalto University

Paths and (k, l)-Loops

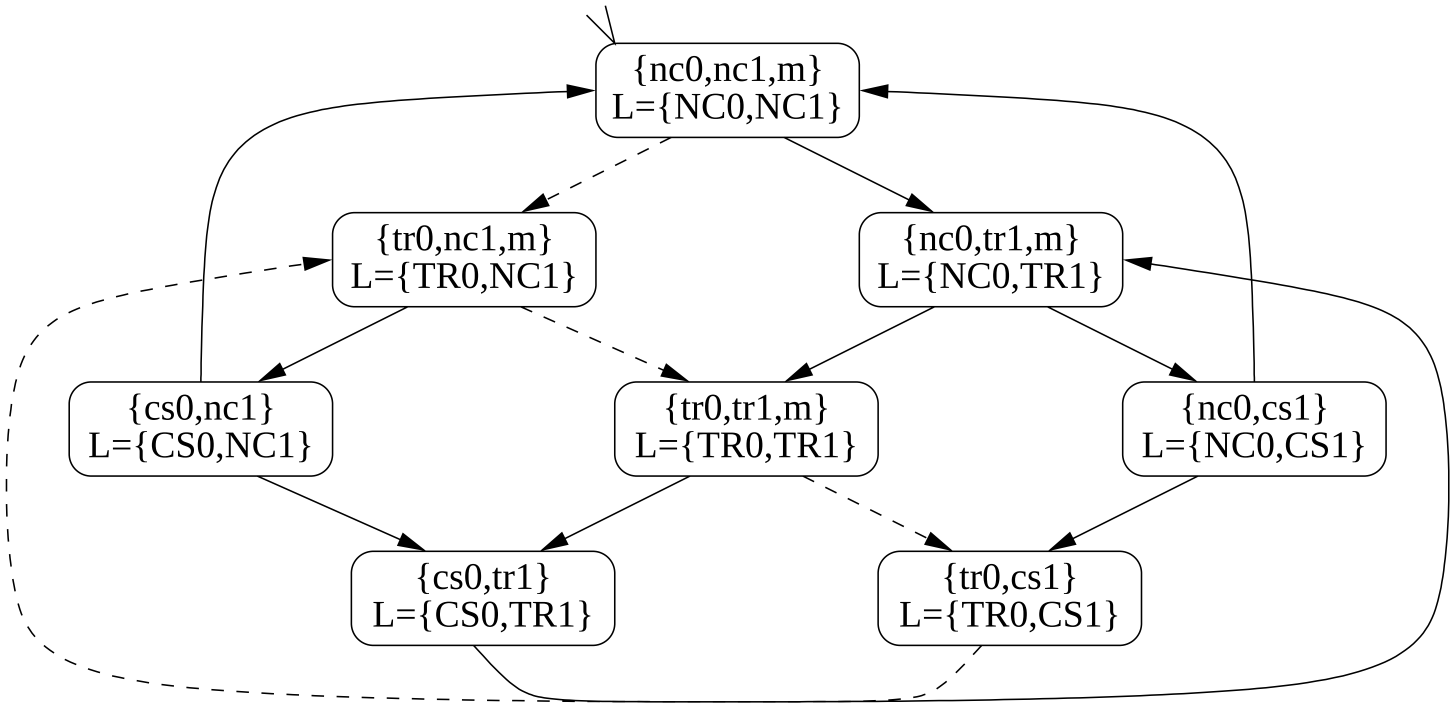
- A path in a Kripke structure $M = (S, s_{init}, T, L)$ is an infinite sequence $\pi = s_0 s_1 \ldots$ of states in S such that
  - $s_0 = s_{init}$, and
  - $T(s_i, s_{i+1})$ holds for all $i \geq 0$
- A path $\pi = s_0 s_1 \ldots$ is a (k, l)-loop if $\pi = (s_0 s_1 \ldots s_{l-1})(s_l \ldots s_k)^{\omega}$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

Aalto University

Running Example: Paths

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

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



Aalto University

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:
  $\mathbf{G}\,\neg(\mathrm{CS0} \wedge \mathrm{CS1})$
- 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:
  $(\mathbf{G}\,\mathbf{F}\,(\mathrm{TR0})) \Rightarrow (\mathbf{G}\,\mathbf{F}\,(\mathrm{CS0}))$

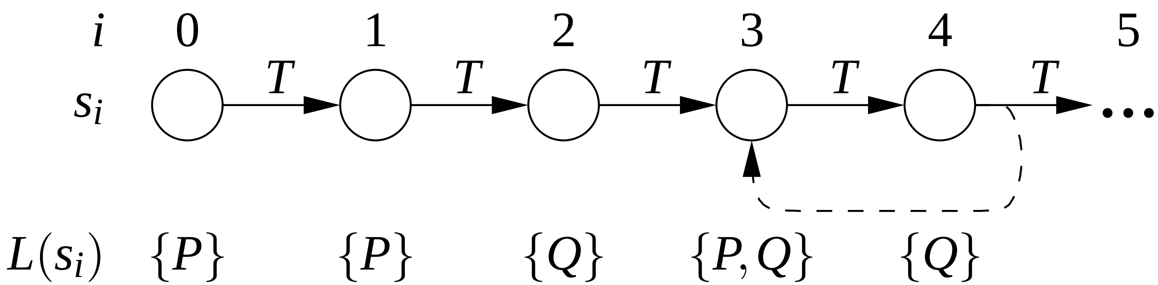
Semantics of LTL

- 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":

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

Aalto University

Semantics of LTL



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

Aalto University

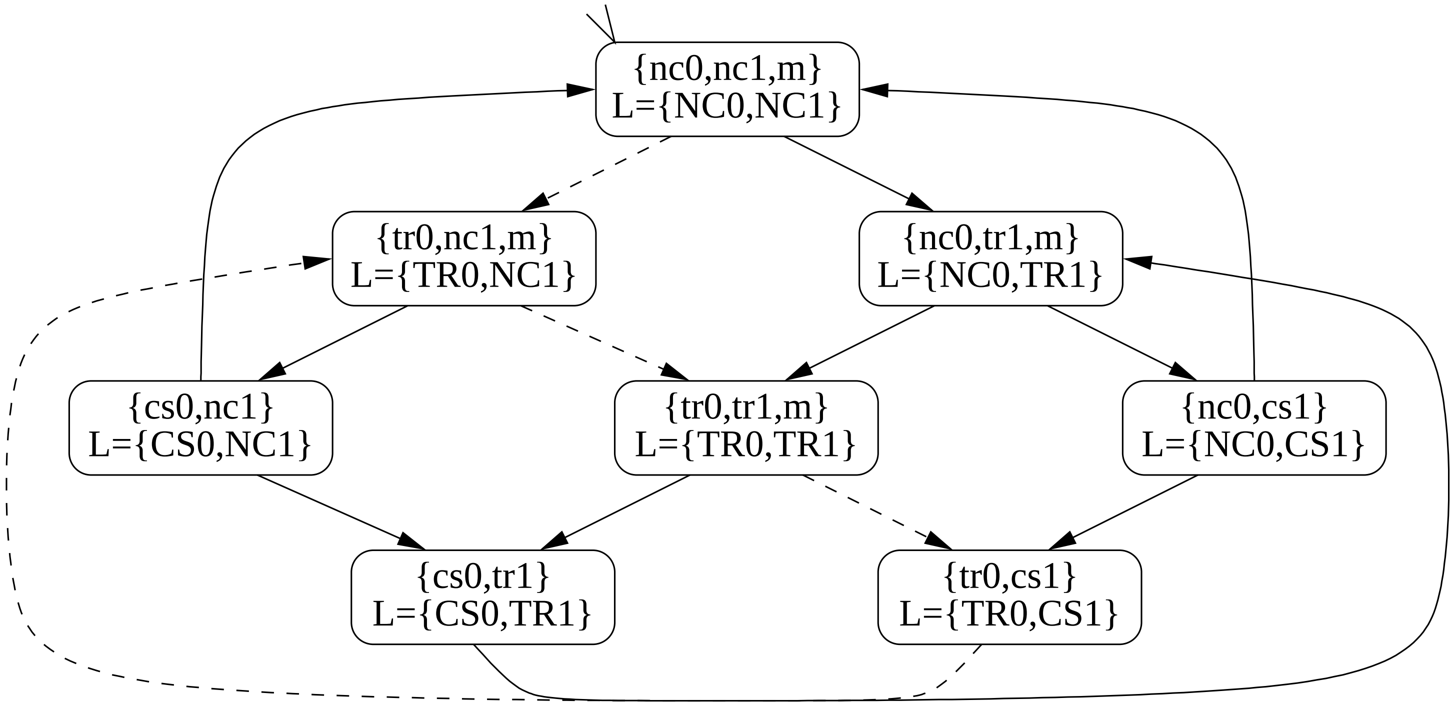
Semantics of LTL

- 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, $M \models \psi$.

Aalto University

Running Example: LTL

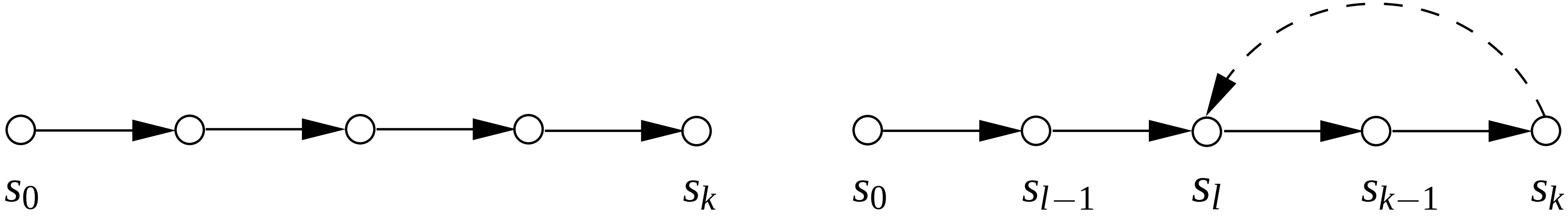
- The dashed path below is a witness for $\mathbf{G}\,(\neg \mathrm{CS0})$ and thus a counter-example for $\neg \mathbf{G}\,(\neg \mathrm{CS0}) \equiv \mathbf{F}\,(\mathrm{CS0})$



Aalto University

Bounded Paths

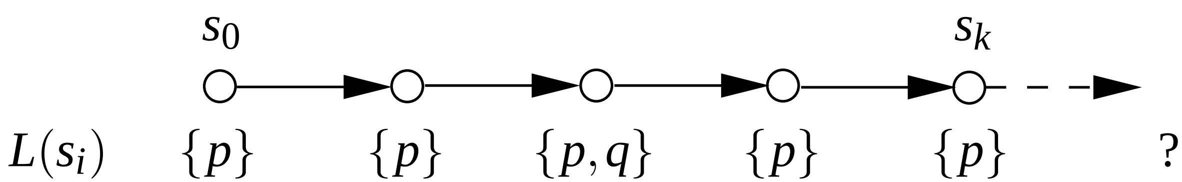
- BMC considers k-paths, i.e., bounded paths with k transitions
- A k-path can represent
  - all its infinite extensions (the "no loop" case), or
  - a (k,l)-loop $s_0 \ldots s_{l-1}(s_l \ldots s_k)^\omega$ if $s_k = s_{l-1}$ for some $1 \leq l \leq k$



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

Aalto University

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 $\pi \models \mathbf{F}\, q$ 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

Aalto University

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:

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

Aalto University

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:

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

Aalto University

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 $\pi \models_{k+1} \psi$.
- The $\models_{nl}$ semantics corresponds to the informative safety counterexamples as defined in:
  Kupferman, O. and Vardi, M. Y.: Model Checking of Safety Properties. Formal Methods in System Design 19(3): 291-314 (2001)

Aalto University

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 $\pi \models_k \psi$

Aalto University

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

Aalto University

Our Approach: Simple BMC

- Heljanko, K., Junttila, T., and Latvala, T.: *Incremental and Complete Bounded Model Checking for Full PLTL*. CAV'05.
  - Incremental and complete version of the encoding for LTL with past time operators
- Biere, A., Heljanko, K., Junttila, T., Latvala, T., and Schuppan, V.: *Linear Encodings of Bounded LTL Model Checking*. Logical Methods in Computer Science 2(5:5):1-64, 2006.
  - Survey of linear LTL encodings for BMC, including also approaches based on Büchi automata based LTL model checking

Aalto University

BMC for LTL: Some Related Work

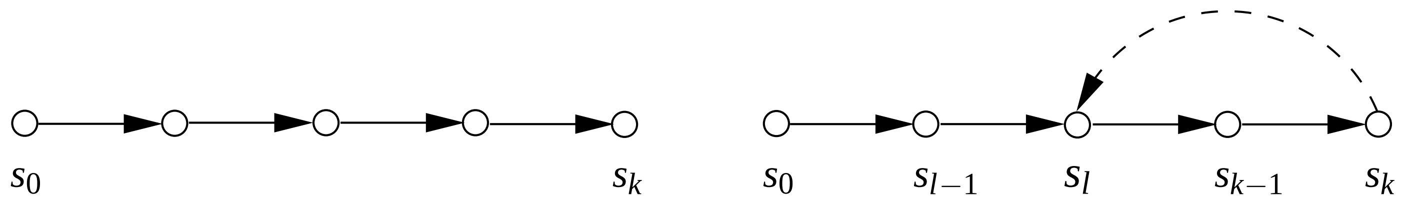
- Biere, A., Cimatti, A., Clarke, E., and Zhu, Y.: Symbolic Model Checking without BDDs. TACAS'99.
  - First LTL to SAT encoding
- Cimatti, A., Pistore, M., Roveri, M., and Sebastiani, R.: Improving the encoding of LTL model checking into SAT. VMCAI'02.
  - Improvements to the above encoding

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



(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: $\Omega(|I| + k \cdot |T| + k^2 \cdot |\psi|)$

Aalto University

Simple BMC Encoding for LTL

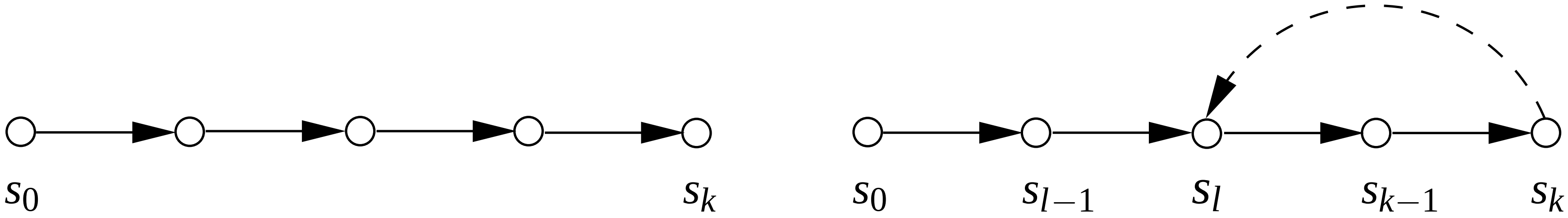
- Goal: build a formula $|[M, \psi, k]|$ that is satisfiable iff M has a path π such that $\pi \models_k \psi$
- The generic form of our translation is

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

- 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 $O(|I| + k \cdot |T| + k \cdot |\psi|)$

Aalto University

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

	$|[\text{LoopConstraints}]|_k$
Base	$l_0 \Leftrightarrow \bot$
	$\text{InLoop}_0 \Leftrightarrow \bot$
$1 \leq i \leq k$	$l_i \Rightarrow (s_{i-1} = s_k)$
	$\text{InLoop}_i \Leftrightarrow \text{InLoop}_{i-1} \vee l_i$
	$l_i \Rightarrow \neg\text{InLoop}_{i-1}$
	$\text{LoopExists} \Leftrightarrow \text{InLoop}_k$

- 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

Aalto University

Illustration of the Encoding

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

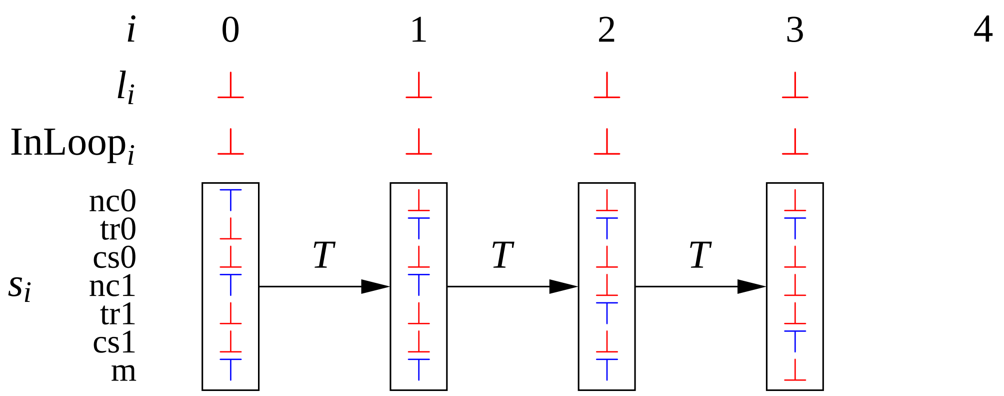
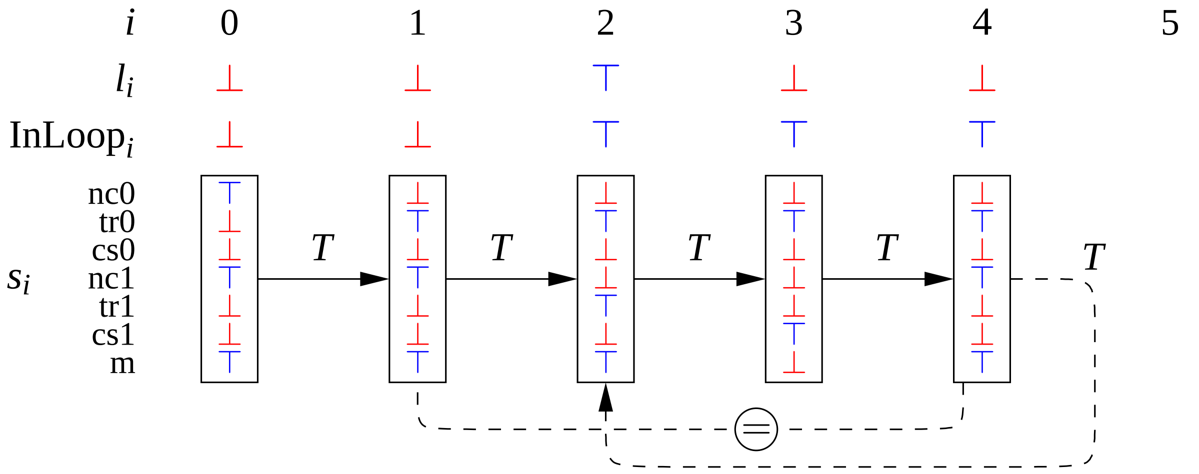


Illustration of the Encoding

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



Encoding LTL: Subformula Variables

- For each subformula φ of ψ, introduce a variable $|[\varphi]|_i$ where $i \in \{0, 1, \ldots, k, k+1\}$
- $|[\varphi]|_i$ evaluates the value of the subformula φ at time step i
- Thus $|[\psi]|_0$ 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

Aalto University

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

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

Aalto University

Encoding LTL Operators (1/4)

- Encoding propositional operators is straightforward

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

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

Aalto University

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

Aalto University

Encoding LTL Operators (4/4)

- Force cyclic dependencies to evaluate correctly

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

Similar to using Büchi automata acceptance sets for ensuring the correct semantics of until formulas on infinite words.

Aalto University

BMC and Incremental SAT Solving

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

Aalto University

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:
  - $\alpha \wedge \beta_0 \wedge \beta_1 \wedge \beta_2 \wedge \gamma_2$
  - $\alpha \wedge \beta_0 \wedge \beta_1 \wedge \beta_2 \wedge \beta_3 \wedge \gamma_3$

Aalto University

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

Aalto University

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

Aalto University

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

Aalto University

BMC beyond 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
- Axelsson, R., Heljanko, K., and Lange, M.: Analyzing Context-Free Grammars Using an Incremental SAT Solver. ICALP'08.
  - A BMC procedure to solve bounded problems on context free grammars

Aalto University

BMC for Branching Time

- Wozna, B.: ACTL* properties and 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

Aalto University

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)

Aalto University

BMC by using Extensions of Propositional SAT

- Sorea, M.: Bounded Model Checking for Timed Automata. ENTCS 68(5),2005.
  - BMC for timed automata
- Audemard, G., Bozzano, M., Cimatti, A., and Sebastiani, R.: Verifying Industrial Hybrid Systems with MathSAT. ENTCS 119:17–32,2005.
  - BMC for linear hybrid automata
- Herde, C., Eggers, A., Fränzle, M., and Teige, T.: Analysis of Hybrid Systems Using HySAT. ICONS 2008: 196-201.
  - A bounded model checker HySAT for hybrid systems

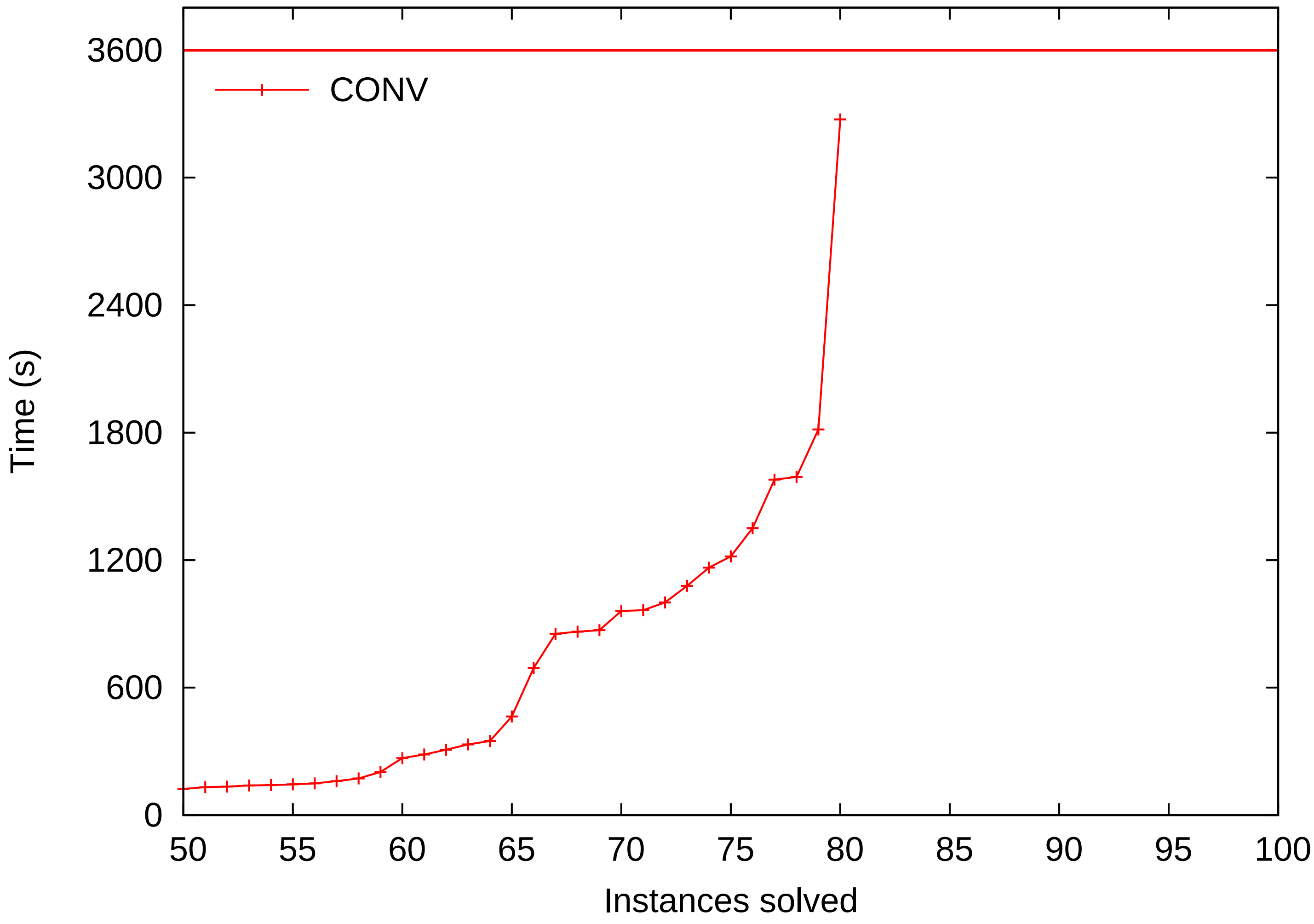
Aalto University

Multicore BMC Engine: Tarmo

- Multicore BMC is an active research topic
- Wieringa, S., Niemenmaa, M., Heljanko, K.: Tarmo: A Framework for Parallelized Bounded Model Checking, In Proceedings of the 8th International Workshop on Parallel and Distributed Methods in Verification (PDMC'09).
- 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

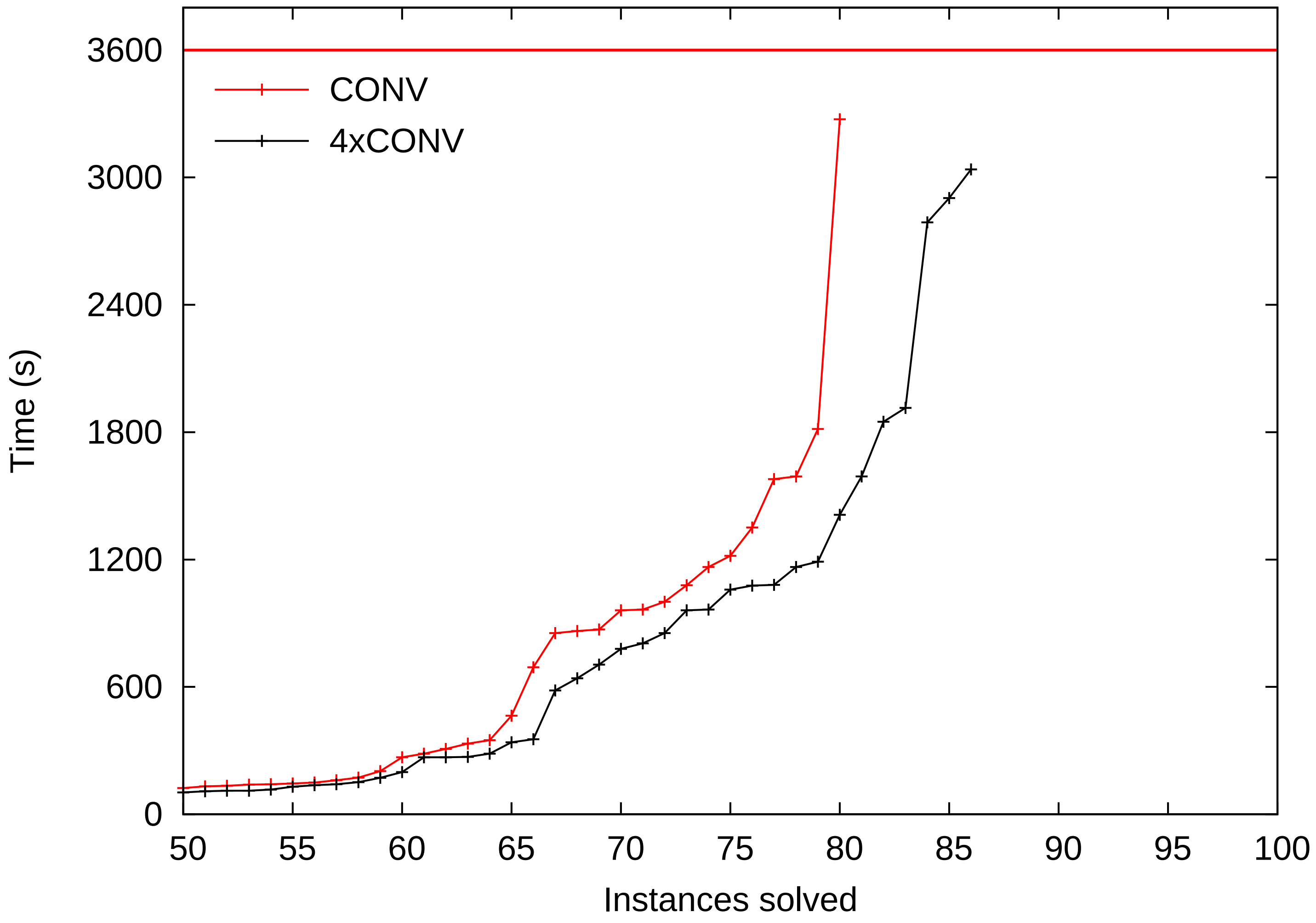
Aalto University

Tarmo: Multicore BMC Experiments



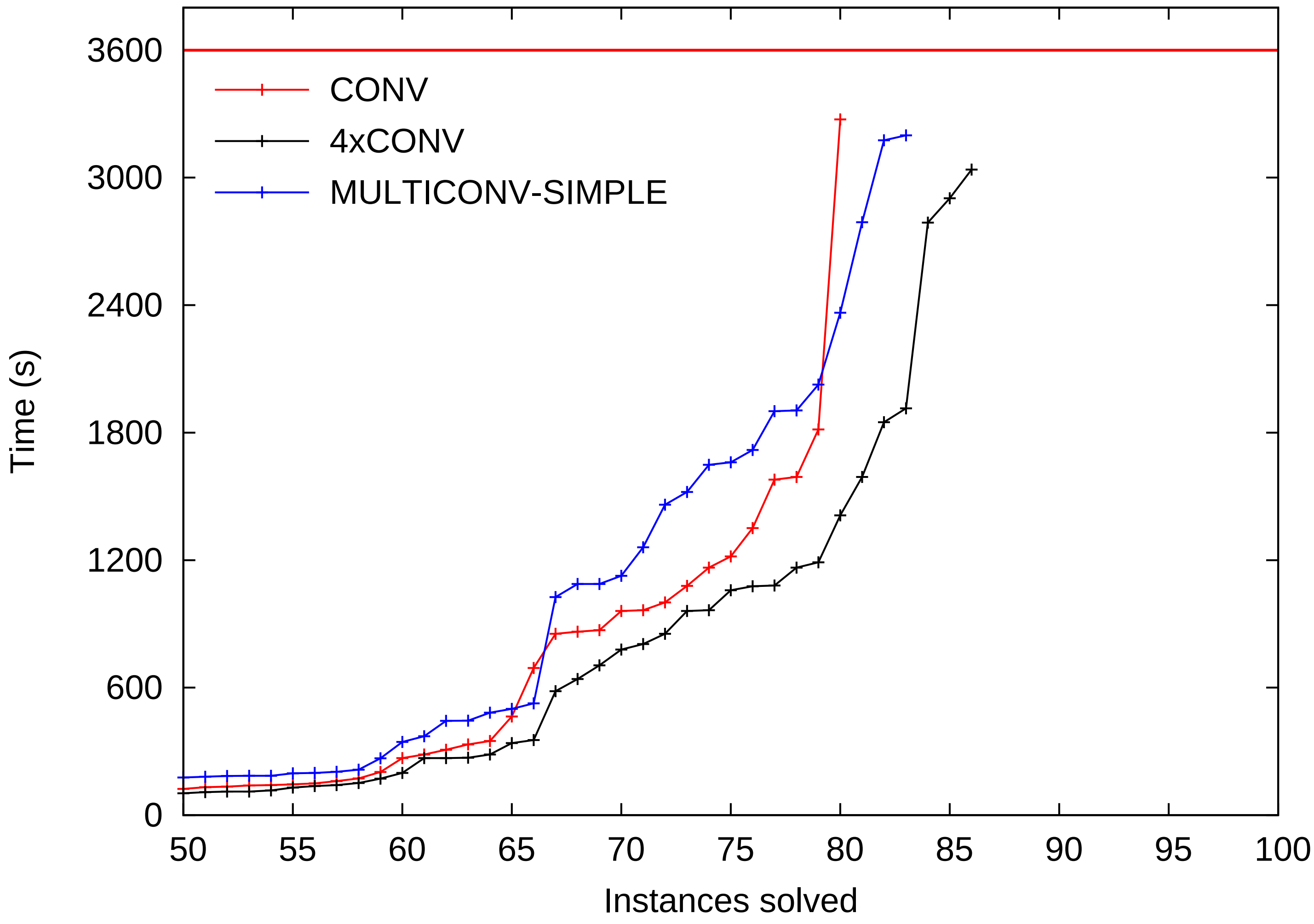
Aalto University

Tarmo: Multicore BMC Experiments



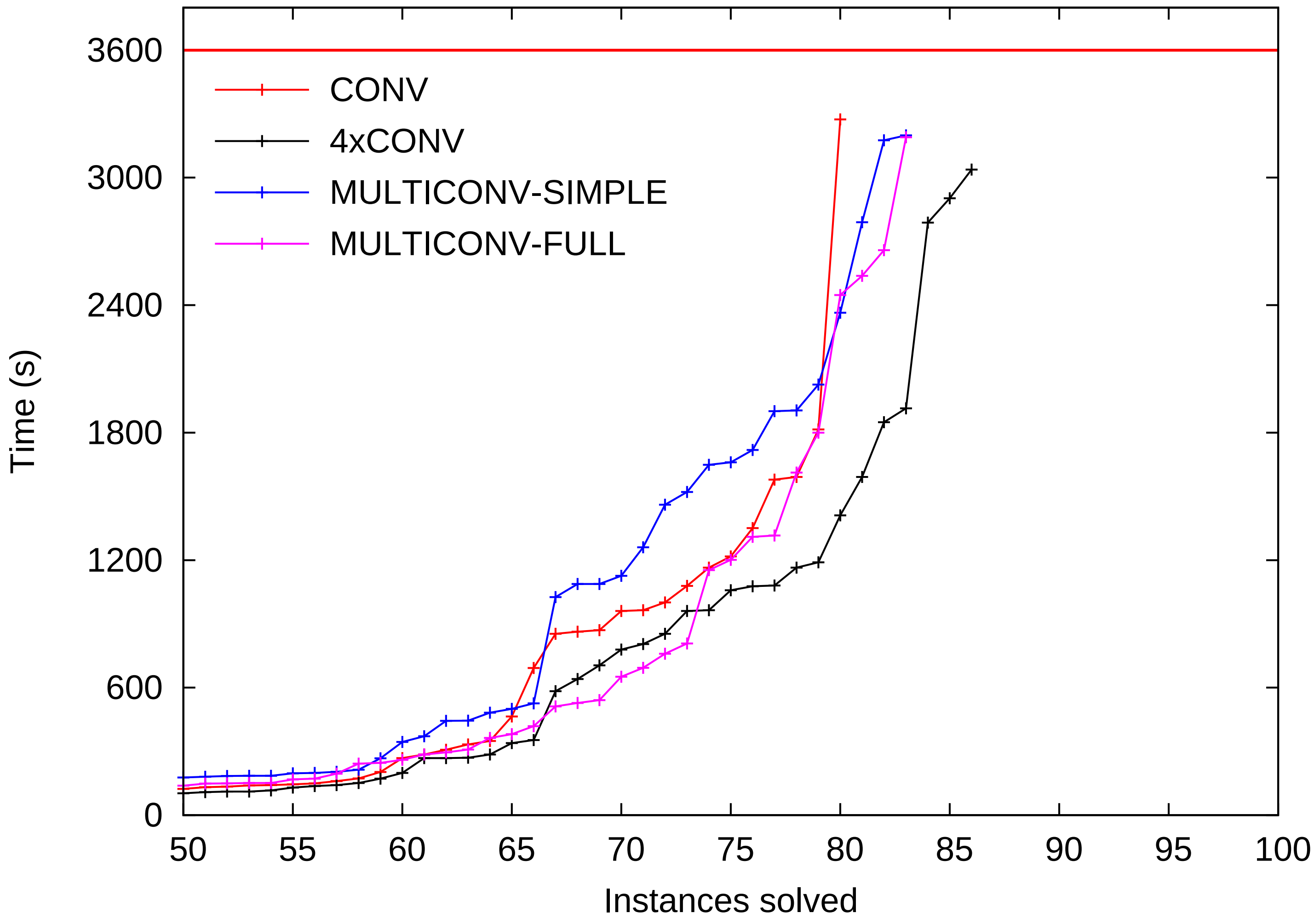
Aalto University

Tarmo: Multicore BMC Experiments



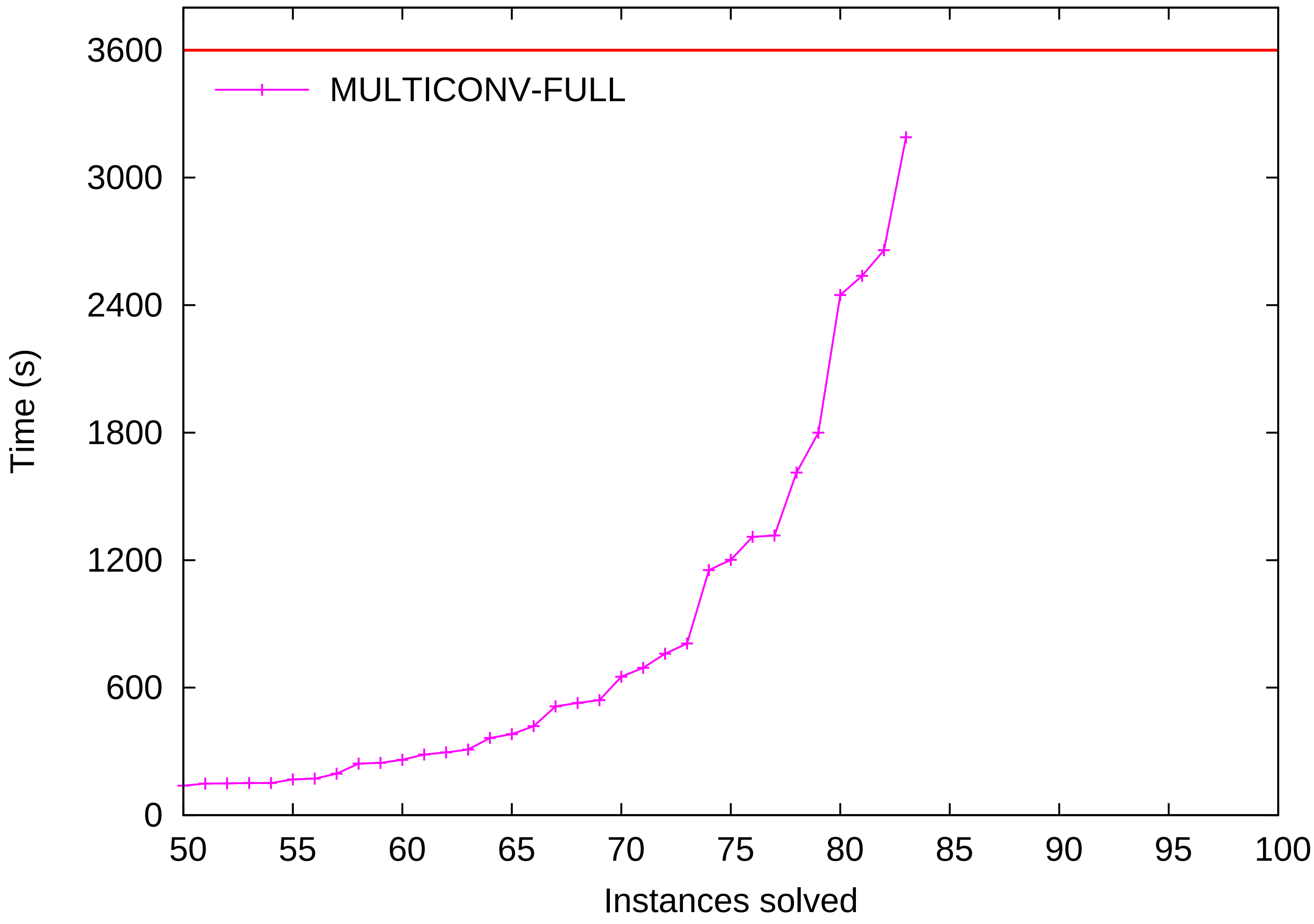
Aalto University

Tarmo: Multicore BMC Experiments



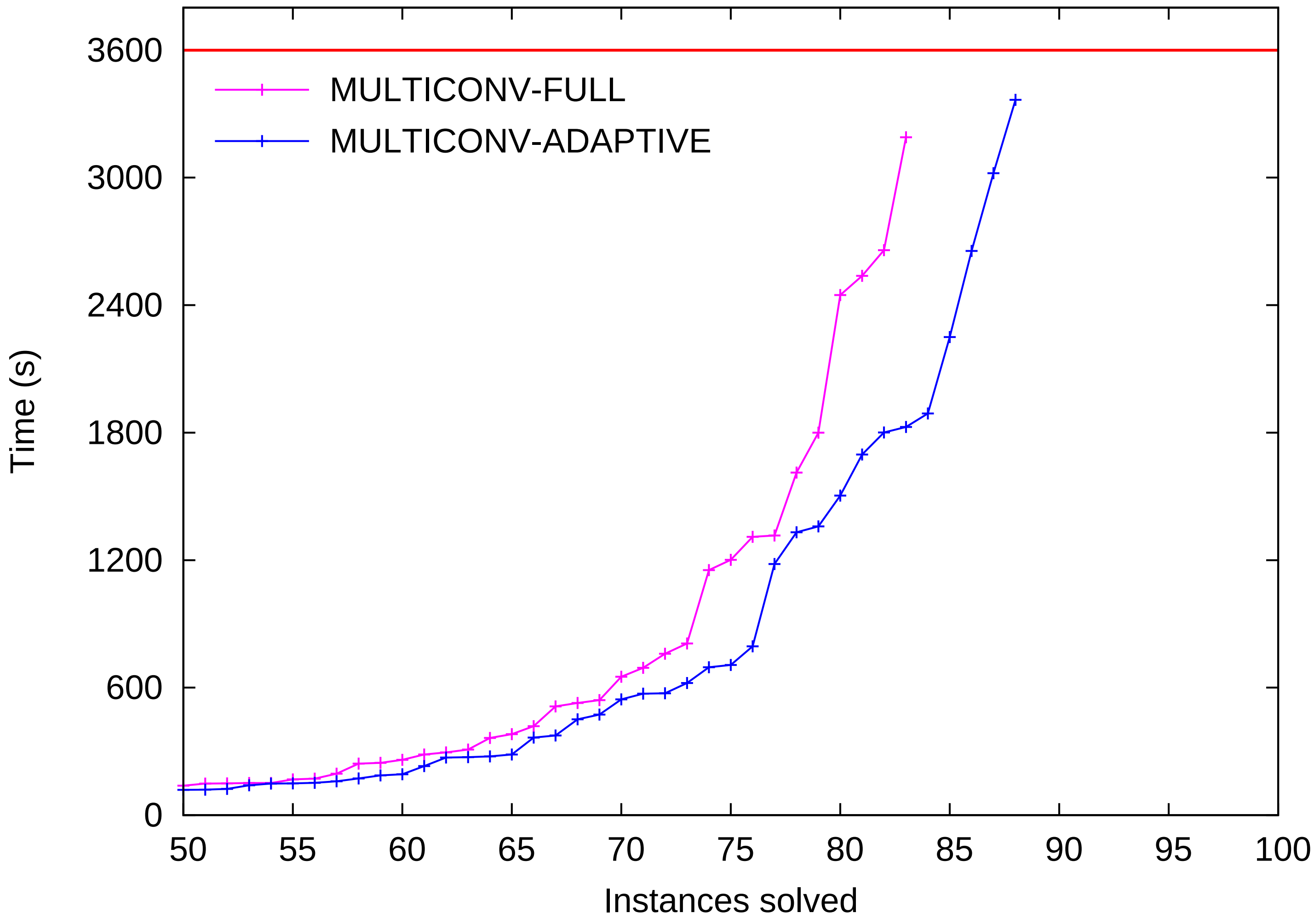
Aalto University

Tarmo: Multicore BMC Experiments



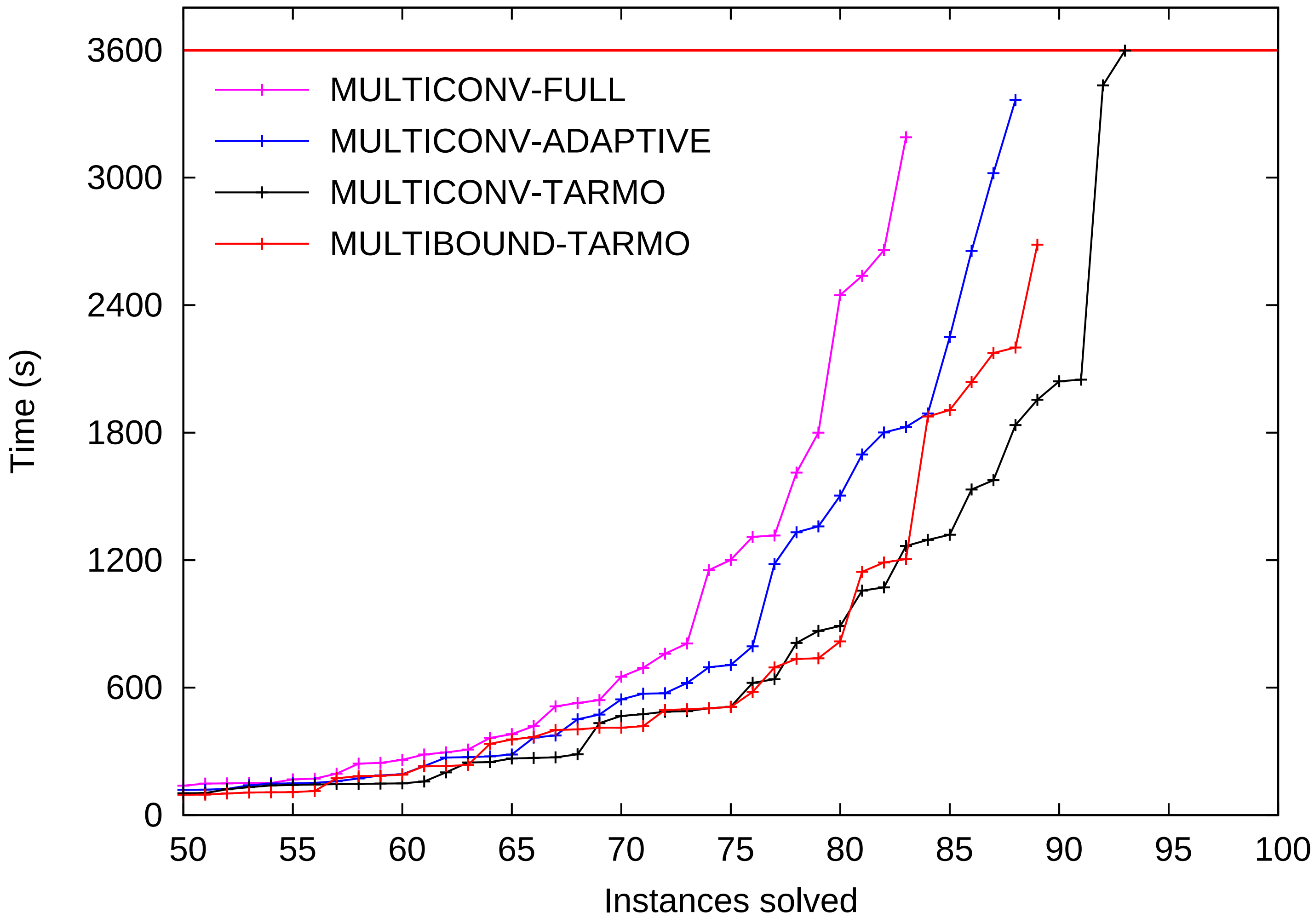
Aalto University

Tarmo: Multicore BMC Experiments



Aalto University

Tarmo: Multicore BMC Experiments



Aalto University

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

Aalto University