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
- Our affiliation: Aalto University, Department of Information and Computer Science

Many thanks to Tommi for letting me use also his slides in preparing this tutorial.

Aalto University

Thanks to co-authors on BMC

- Roland Axelsson & Martin Lange, LMU München
- Armin Biere, Johannes Kepler University of Linz
- Toni Jussila, OneSpin Solutions
- Misa Keinänen, European Batteries
- Timo Latvala, Space Systems Finland
- Ilkka Niemelä, Aalto University
- Matti Niemenmaa, Aalto University
- Jussi Rintanen, National ICT Australia
- Viktor Schuppan, Fondazione Bruno Kessler
- Siert Wieringa, Aalto University

Aalto University

Software failures

Software is used widely in many applications where a bug in the system can cause large damage:

- Safety critical systems: airplane control systems, medical care, train signalling systems, air traffic control, etc.
- Economically critical systems: e-commerce systems, Internet, microprocessors, etc.

Price of Software Defects

Two very expensive software bugs:

- Intel Pentium FDIV bug (1994, approximately $500 million).
- Ariane 5 floating point overflow (1996, approximately $500 million).

Pentium FDIV - Software bug in HW

4195835 - ((4195835 / 3145727) * 3145727) = 256

The floating point division algorithm uses an array of constants with 1066 elements. However, only 1061 elements of the array were correctly initialized.

Aalto University

Ariane 5

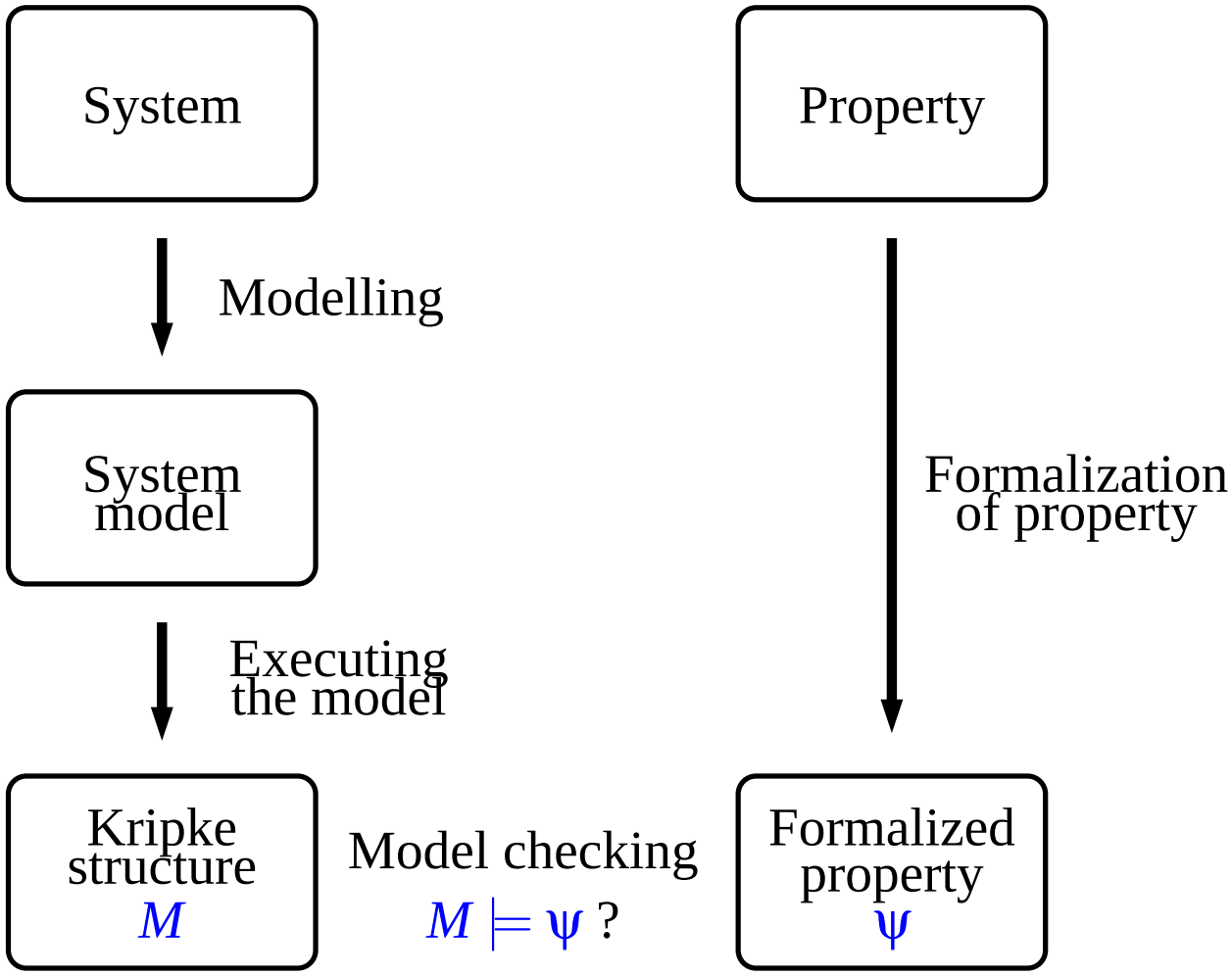
Exploded 37 seconds after takeoff - the reason was an overflow in a conversion of a 64 bit floating point number into a 16 bit integer.

Model Checking

In model checking every execution of the model of the system is simulated obtaining a Kripke structure M describing all its behaviors. M is then checked against a property ψ:

- Yes: The system functions according to the specified property (denoted $M \models \psi$).
  The symbol $\models$ is pronounced “models”,
  hence the term model checking.
- No: The system is incorrect (denoted $M \not\models \psi$), a counterexample is returned: an execution of the system which does not satisfy the property.

Models and Properties



Aalto University

Benefits of Model Checking

- In principle automated: Given a system model and a property, the model checking algorithm is fully automatic.
- Counterexamples are valuable for debugging.
- Already the process of modelling catches a large percentage of the bugs: good for rapid prototyping of concurrency related features.

Drawbacks of Model Checking

- State explosion problem: Capacity limits of model checkers can be exceeded.
- Manual modelling often needed:
  - Model checker used might not support all features of the final implementation language.
  - Abstraction used to overcome capacity problems.

Model Checking in the Industry

- Microprocessor design: Several major microprocessor manufacturers use model checking methods as a part of their design process.
- Mission Critical Software: NASA space program is model checking code used by the space program.
- Operating Systems: Microsoft is using model checking to verify the correct use of locking primitives in Windows device drivers.
- Safety Critical Systems: Model checking is used to find bugs in many safety critical systems

Finite-State Model Checking Tools

- Explicit State Model Checking: Tools include Spin, Murφ Java Pathfinder DiVinE, CADP, etc.
- BDD based Symbolic Model Checking: Tools include NuSMV 2, VIS, Cadence SMV, etc.
- Bounded Model Checking: Tools include NuSMV 2, CMBC, VIS, Cadence SMV, etc.

In addition there are quantitative model checking tools excluded from this list but you will hear about in the next few days.

Bounded Model Checking

- Originally presented in the paper: Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Yunshan Zhu: Symbolic Model Checking without BDDs. TACAS 1999: 193-207, LNCS 1579.
- A closely related approach had already been used earlier to solve artificial intelligence planning problems in: Henry A. Kautz, Bart Selman: Planning as Satisfiability.Proceedings of the 10th European conference on Artificial intelligence (ECAI'92): 359-363, 1992, Kluwer.

Basics of Bounded Model Checking

- The basic idea is the following: Encode all the executions of the system M of length k into a propositional formula $\left|\left[M\right]\right|^k$.
- Conjunct this formula with a formula $\left|\left[\neg\psi\right]\right|^k$ which is satisfiable for all executions the system of length k which violate the property ψ.
- If the formula $\left|\left[M\right]\right|^k \wedge \left|\left[\neg\psi\right]\right|^k$ is satisfiable, a counterexample has been found.
- If the formula $\left|\left[M\right]\right|^k \wedge \left|\left[\neg\psi\right]\right|^k$ is unsatisfiable, no counterexample of length k exists.

SAT

- The propositional satisfiability problem (SAT) is one of the main instances of NP-complete problems.
- Thus no polynomial algorithms for SAT are known.
- However, there are highly efficient SAT solvers available such as zChaff and MiniSAT which are able solve many bounded model checking problems efficiently.

SAT References

- zChaff: Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, Sharad Malik: Chaff: Engineering an Efficient SAT Solver. DAC 2001: 530-535, ACM.
- MiniSAT: Niklas Eén, Niklas Sörensson: An Extensible SAT-solver. SAT 2003: 502-518, LNCS 2919.
- SATLive! - Links to SAT related events, tools, position announcements, etc.

Basic Setup

- For simplicity first consider the following setup:
  - As system models we consider systems whose state vector s consist of n Boolean state variables $\langle s[0], s[1], \ldots, s[n-1] \rangle$.
  - We take $k+1$ copies of the system state vector denoted by $s_0, s_1, \ldots, s_k$.
  - Let $I(s)$ be the initial state predicate of the system, and $T(s, s')$ be the transition relation both expressed as propositional formulas.

A Simplifying Assumption

- For simplicity we assume $T(s, s')$ to be be total for now, i.e., every reachable state s should have a successor s' such that $T(s, s')$ holds.

Unrolling the Transition Relation

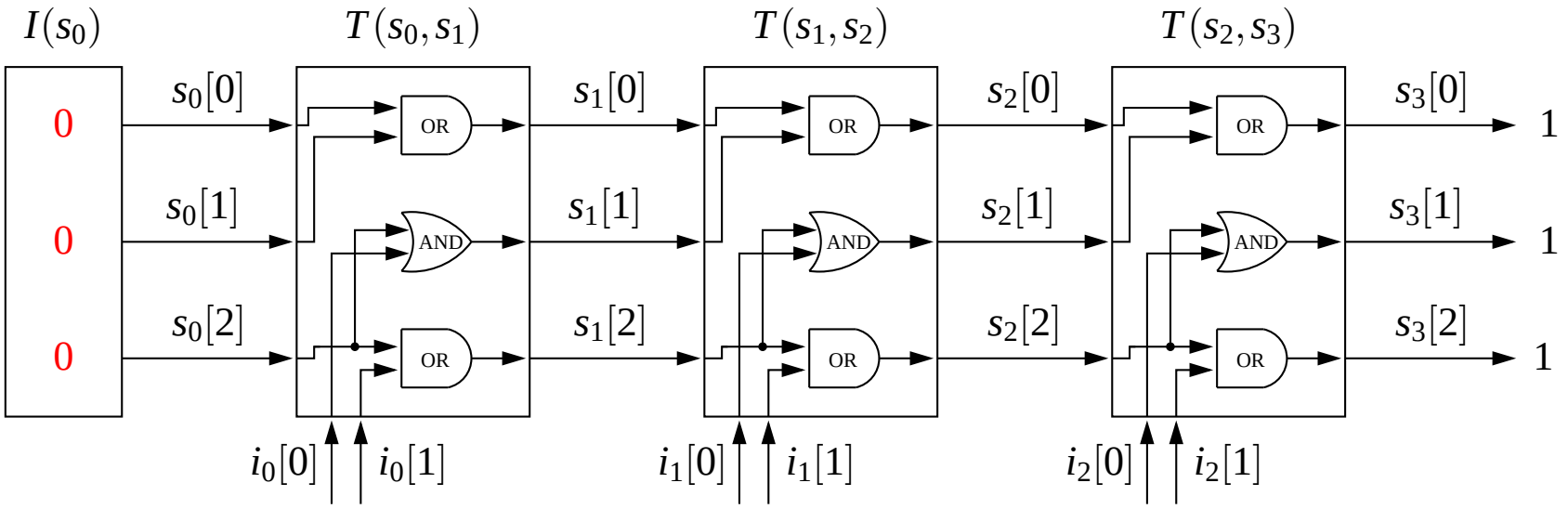
- The executions of the system of length k are captured by the formula:

$$|[M]|^k = I(s_0) \wedge \bigwedge_{i=1}^{k} T(s_{i-1}, s_i)$$

- For $k = 3$ this becomes:

$$|[M]|^3 = I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3)$$

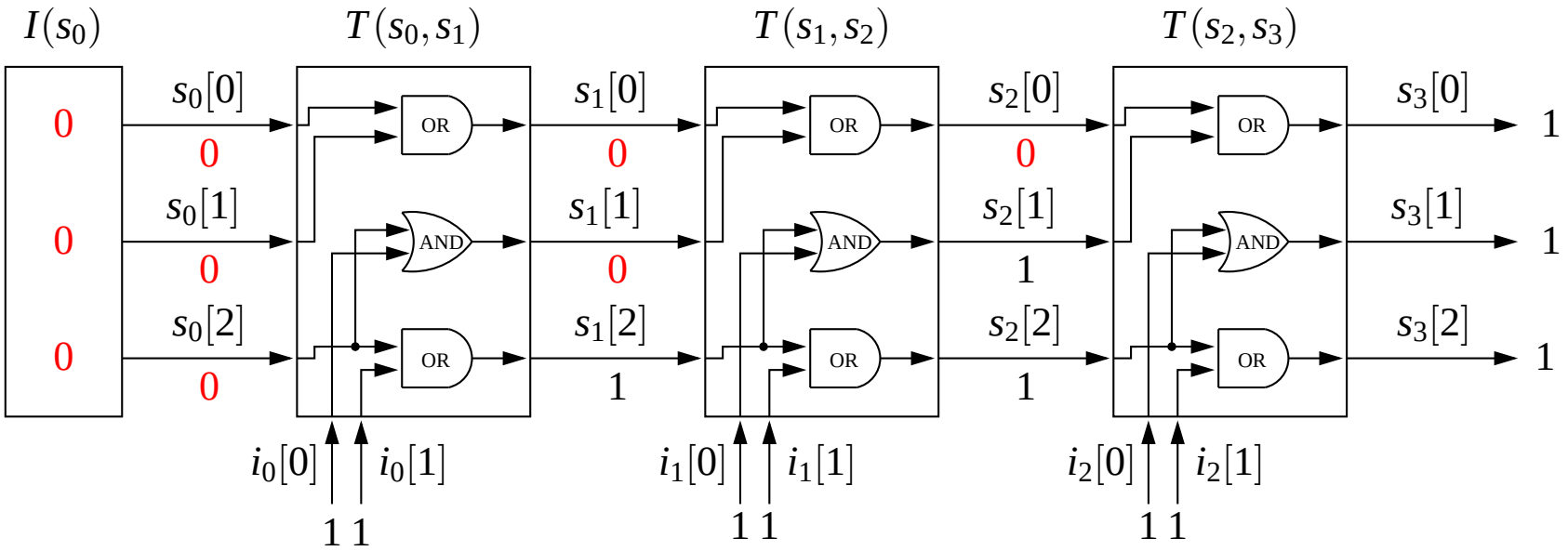
Circuit BMC Unrolling



What do the input vectors i_0, i_1, and i_2 need to be to reach the state $s_3 = \langle 1, 1, 1 \rangle$?

Aalto University

Circuit BMC Unrolling Solution



The input vectors $i_0 = \langle 1, 1 \rangle$, $i_1 = \langle 1, 1 \rangle$, and $i_2 = \langle 1, 1 \rangle$ will reach the final state $s_3 = \langle 1, 1, 1 \rangle$.

Expressing Invariants

- Suppose the property ψ we want to model check is that an invariant property $P(s)$ holds for every reachable state of the system M.
- Now we get that:

$$||[\neg\psi]||^k = \bigvee_{i=0}^{k} \neg P(s_i)$$

- Thus for $k = 3$ this becomes:

$$||[\neg\psi]||^3 = \neg P(s_0) \vee \neg P(s_1) \vee \neg P(s_2) \vee \neg P(s_3)$$

Final formula

- Thus the final formula $|[M]|^k \wedge |[\neg\psi]|^k$ for $k = 3$ becomes:

$$I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge T(s_2, s_3) \wedge$$
$$(\neg P(s_0) \vee \neg P(s_1) \vee \neg P(s_2) \vee \neg P(s_3))$$

- If the formula is satisfiable, then an execution of the system of length 3 exists which violates the invariant property $P(s)$ in some state during the execution.

Reachability Diameter

- If the formula is unsatisfiable, we have proved that there is no execution of length at most 3 that violates the invariant.
- Clearly for every finite state system there is some bound d called the reachability diameter such that from the initial state every reachable state is reachable with an execution of at most length d.
- By taking $d = 2^n$, where n is the number of state bits, we could guarantee completeness.
- Unfortunately computing better approximations of d are computationally hard in the general case.

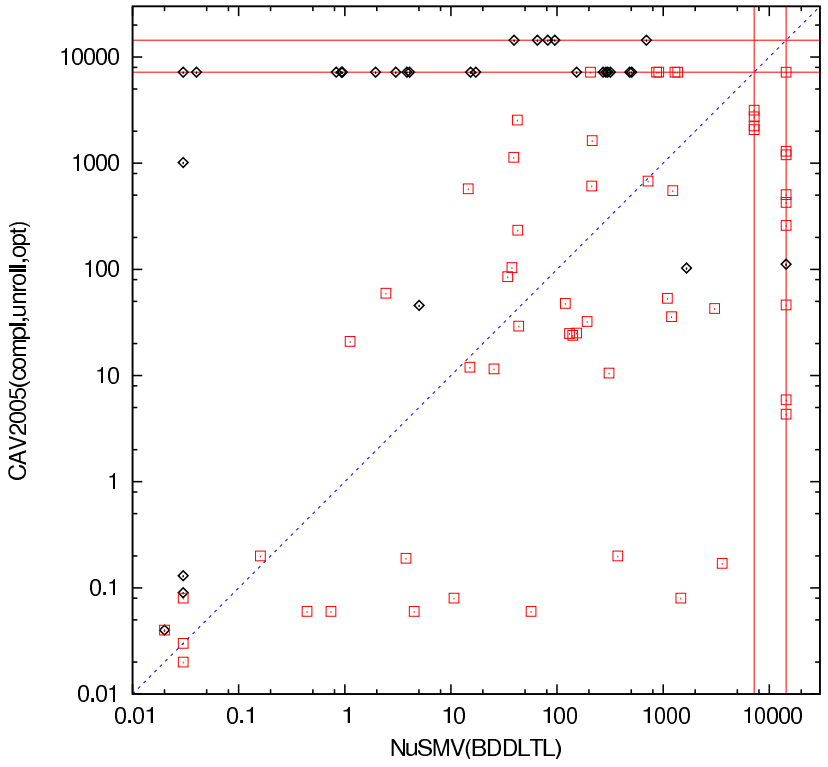
Unsatisfiable - Increase the bound

- Unfortunately the approach of taking $d = 2^n$ is not viable for anything but trivially small systems.
- Usually d is only increased by a small amount, say 1, and the procedure is repeated from the beginning until some resource limit (running time, memory, etc.) is hit.

BMC: Pros and Cons

- Boolean formulas can be more compact than BDDs
- Leverages efficient SAT-solver technology
- Minimal length counterexamples (often, not always)
- Basic method is incomplete
- Not always better than BDD-based methods or explicit state model checking

BDDs vs. BMC on Hardware Designs



- Runtimes of NuSMV 2.2.3/BDDLTL and NuSMV 2.2.3+our BMC engine presented in CAV05 on synchronous HW designs with PLTL properties

Aalto University

Alternative Transition Relations

- When checking for reachability properties such as the violation of invariants, in asynchronous systems we can often replace the transition relation $T(s, s')$ with an alternative transition relation definition $T'(s, s')$ provided that:
  - Every state that is reachable from the initial state s_0 using $T(s, s')$ must be reachable from s_0 using $T'(s, s')$.
  - There should not be any new states reachable from s_0 using $T'(s, s')$ which are not reachable from s_0 using $T(s, s')$.

Encoding the Transition Relation

- There are now in fact many different ways to pick and encode an alternative transition relation $T'(s, s')$ if we consider asynchronous systems containing concurrency.
- The earliest paper to consider alternative transition relations in BMC is:
  Keijo Heljanko: Bounded Reachability Checking with Process Semantics. CONCUR 2001: 218-232, LNCS 2154.

A Wish-List for Encodings

- A wish-list of mutually conflicting requirements for $T'(s, s')$ and its encoding:
  - Compact, hopefully linear encoding size in the size of the system description.
  - Covers as many reachable states as possible for each bound k without losing soundness or completeness.
  - Efficiently solvable by the SAT solver.

Transition Relation Encoding

- Note that in the list of requirements we do not explicitly list that the number of state variables n should be minimized.
- This is often one of the main things to optimize with a BDD based symbolic model checker.
- Having too compact an encoding of the state vector can lead to losses in the SAT solver efficiency!
- More research is needed on how to more efficiently encode transition relations for different classes of systems. There are dramatic performance differences, at least for asynchronous systems.

Alternative Transition Relations

- Next we describe alternative transition relations for synchronization of LTSs.
- The encoding has been published in:
  Toni Jussila, Keijo Heljanko, Ilkka Niemelä:
  BMC via on-the-fly determinization. STTT 7(2):
  89-101 (2005).

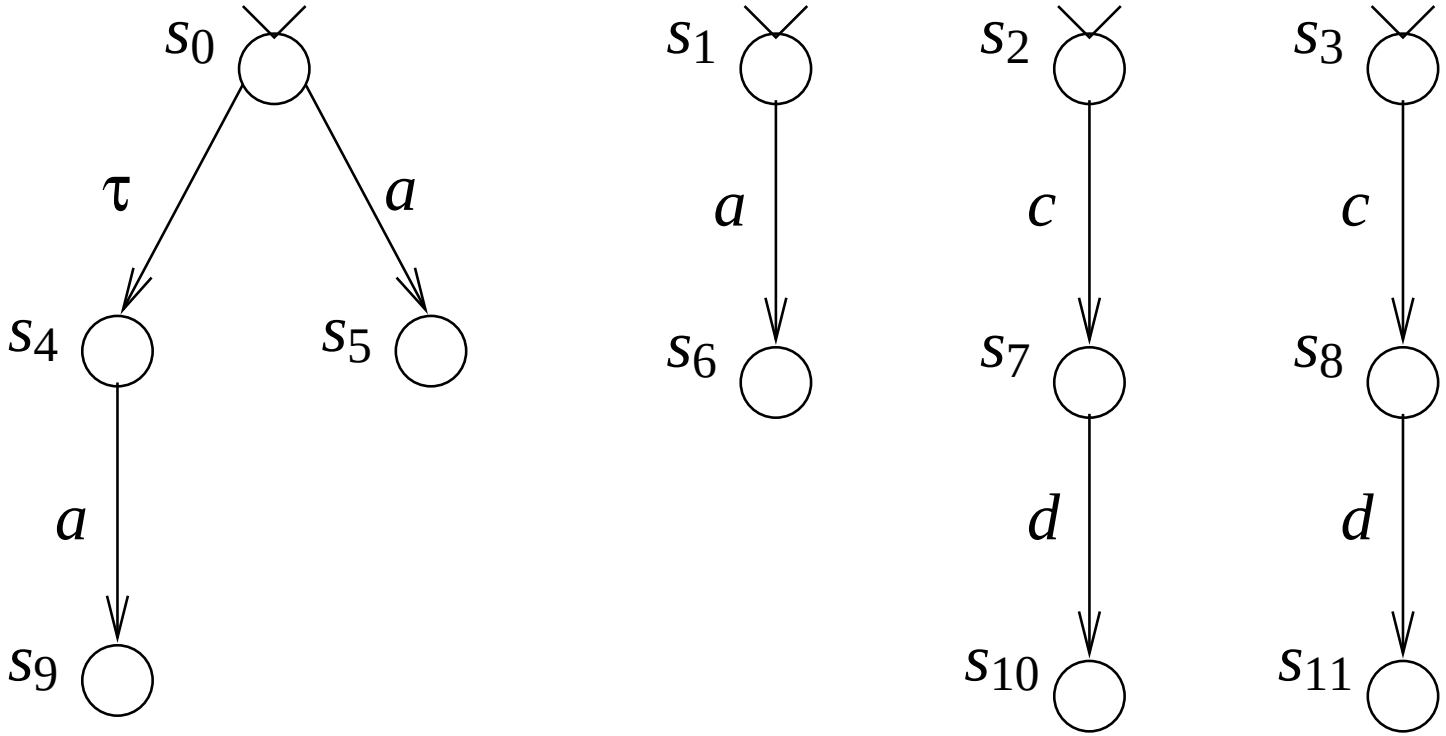
Intuition: LTS Semantics

- We use the standard synchronization construction for LTSs (see the paper mentioned in the previous slide for details): The system consists of n LTSs $L_1, L_2, \ldots, L_n$ composed as $L = L_1 \,||\, L_2 \,||\, \cdots \,||\, L_n$.
- Each LTS has its own alphabet. The system L can make a move with a letter a iff every LTS with a in its alphabet is able to perform it.
- When a is performed, every LTS with a in its alphabet moves, while the others do not change their state.
- In addition, each LTS can make local τ-labelled moves on their own.

Alternative Semantics

- Next we show by using a running example what the state spaces induced by the presented alternative semantics for LTSs are.
- Thanks to Toni Jussila for allowing the use of Figures from his Thesis in the following slides.
  Toni Jussila.
  On bounded model checking of asynchronous systems. Research Report A97, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, October 2005. Doctoral dissertation.

LTSs: Running Example



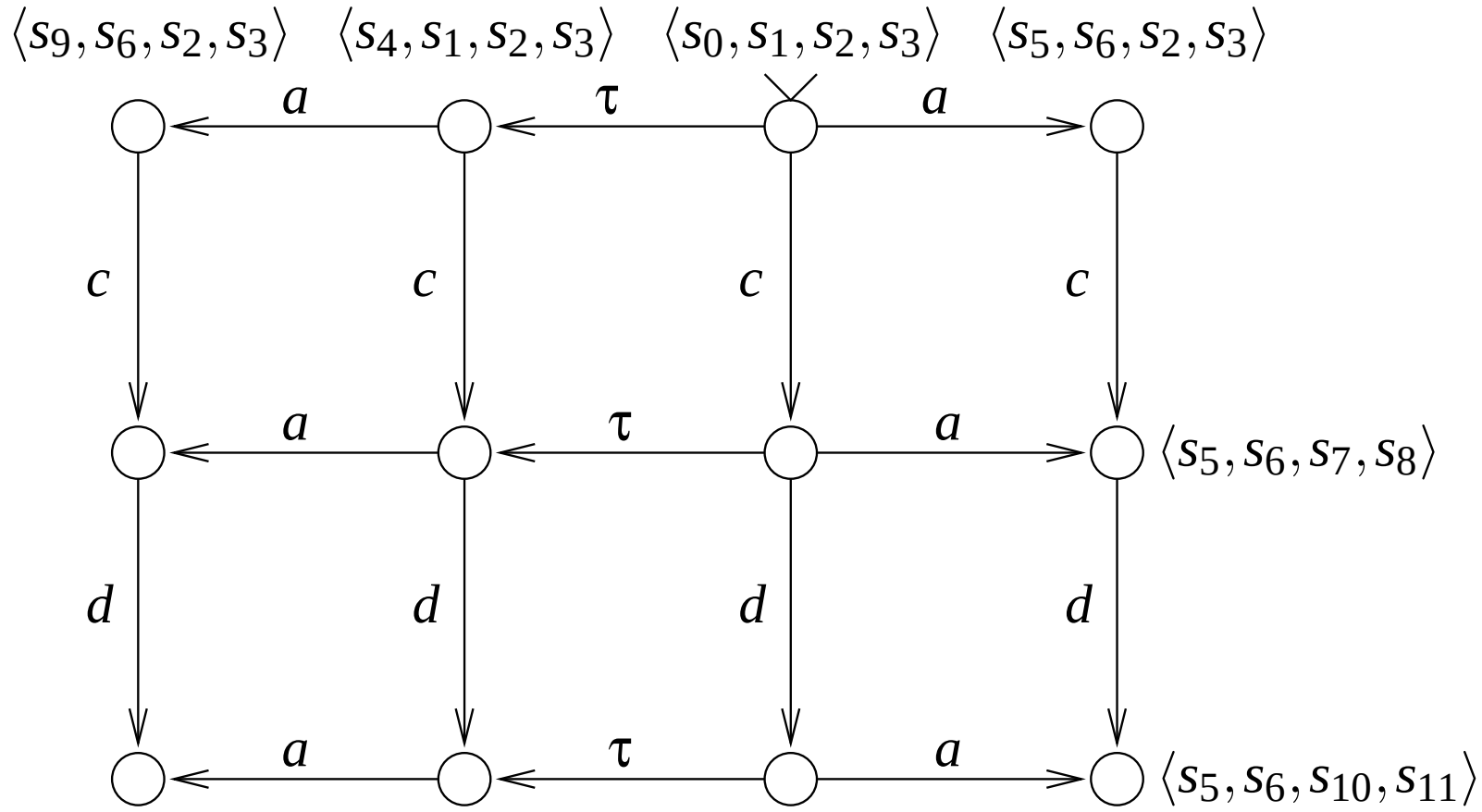
$\Sigma_1 = \Sigma_2 = \{a\}, \Sigma_3 = \Sigma_4 = \{c, d\}$

The complete system is $L = L_1 || L_2 || L_3 || L_4$.

Aalto University

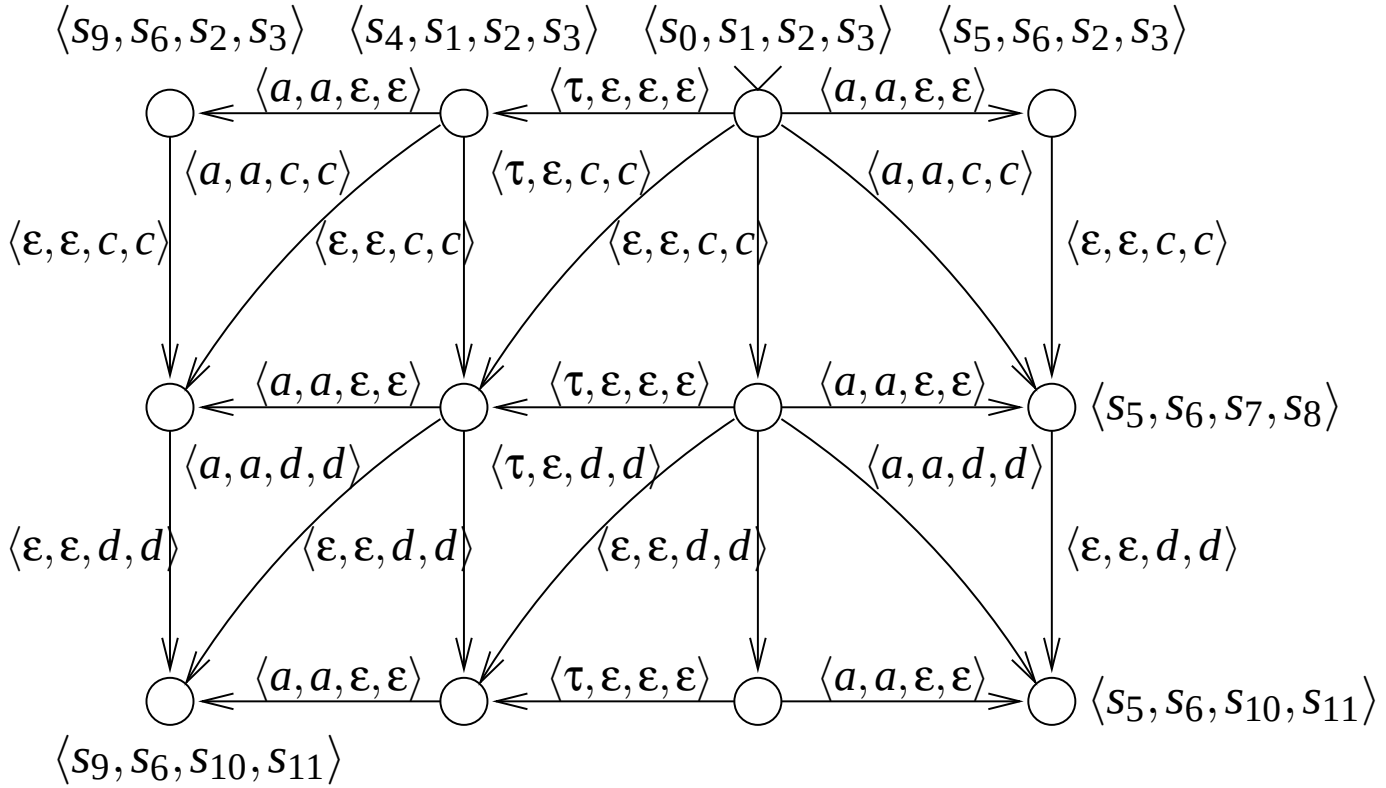
LTSs: Interleaving Semantics

The (interleaving) reachability graph is as expected:



LTSs: Step Semantics

In step reachability graph a set of synchronizations are independent if they occur in disjoint sets of LTSs. Such sets can be concurrently executed:



Aalto University

Properties Steps Semantics

- The transition relation for steps can be represented as: $T(s_i, a_i, s_{i+1})$, where a_i is the set of actions fired at time i, modeled as free input variables.
- Because all singleton sets are also steps, the (interleaving) reachability graph is always a subgraph of the step reachability graph.
- Because the final state reached after firing a step is the final state of every interleaving of the step, no new reachable states have been introduced.
- The reachability diameter of the system is in the worst case as big as in the interleaving case.

Interleaving vs. Steps

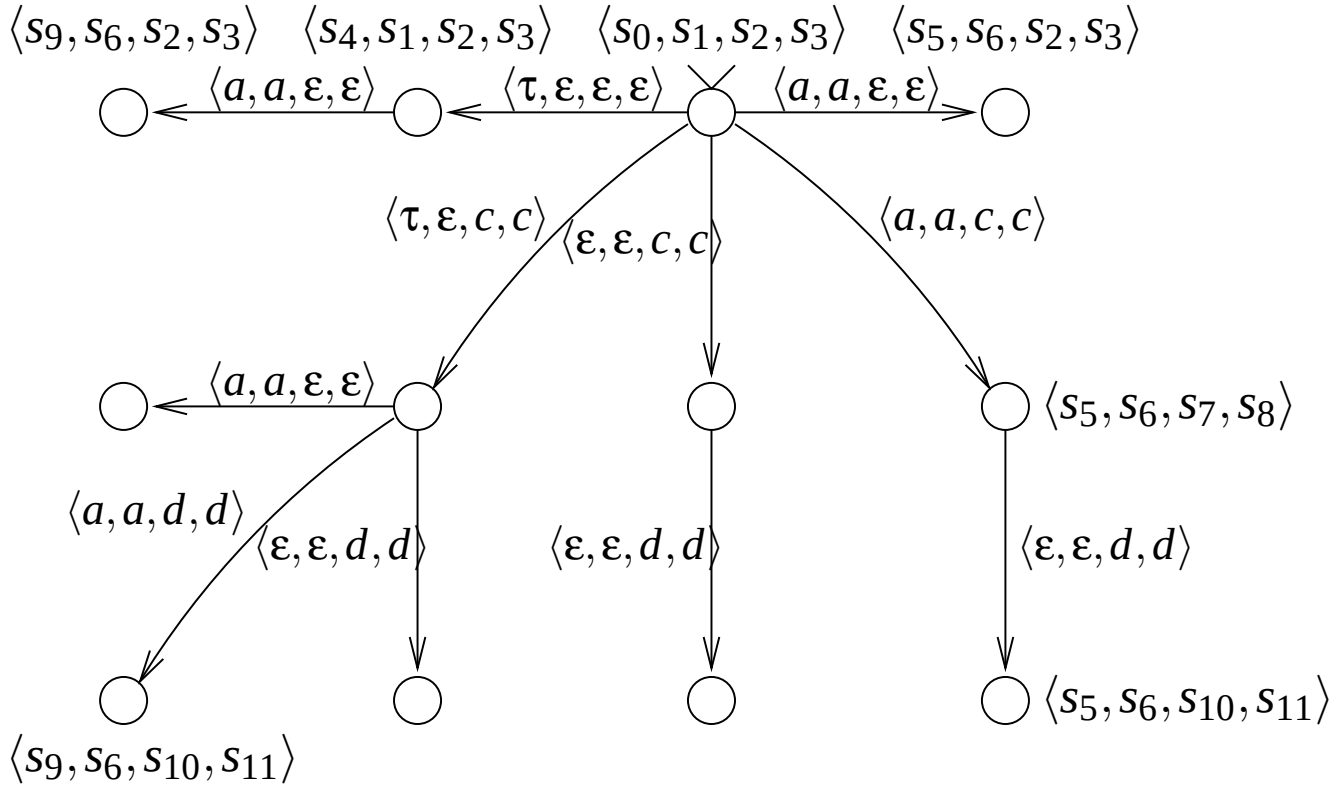
- Step transition relation can be encoded without extra blowup, often counterexamples are found with smaller bounds using less SAT solver time.
- Quite often even small reductions in the required bound translate to large performance differences.
- The step encoding also is more “local” than the interleaving encoding:
- We have not yet found a domain where the interleaving encoding would be superior in performance to the step encoding.

Process Semantics

- The step executions introduce a lot of edges to the step reachability graph that allow states to be reached with many different step executions.
- Can we somehow pick a unique canonical representative of such “concurrent” behavior, and thus reduce the number of different executions the SAT solver has to consider?
- The answer turns out to be positive. The resulting semantics will be called process semantics.
- There is even a compact (linear size) SAT encoding to capture the process semantics.

LTSs: Process Semantics

In the process semantics a synchronization can happen at step S_i only if at least one participant of it was active at step S_{i-1}:



Aalto University

Properties of Processes

- The transition relation for processes can be represented as: $T(s_i, a_{i-1}, a_i, s_{i+1})$, where a_i is the set of actions fired at time i and a_{i-1} is the set of actions fired at time $i-1$.
- Each state of the system is reachable by a process execution that is among the shortest step executions to reach that state.
- Furthermore, the process reachability diameter is always as small as the step reachability diameter.
- There are at most as many process executions as there are interleaving executions of length k.

Steps vs. Processes

- There can be exponentially more step and interleaving executions of length k than there are process executions.
- The processes executions are basically the Foata normal form from the theory of Mazurkiewicz traces.
- Processes can be seen as “the optimal partial order reduction method”: Each partial order execution has exactly one representative.
- Unfortunately there is some bad news: processes are ofter slower than steps with SAT solvers incorporating learning.

Experiments: Different Semantics

Table 7.1: Test Results of Interleaving, Step and Process Models

Problem	Il. k	Il. s	St. k	St. t	Pr. k	Pr. s
Dac(200)	3	**0.51**	3	0.98	3	0.87
Dartes	32	**2.7**	32	3.4	32	4.0
Dp(12)	12	$> 1h$	1	**0.02**	1	**0.02**
Elev(4)	17	268	10	4.0	10	**3.1**
Hart.(100)	201	97.7	201	**81.2**	201	147
Key(4)	50	962	37	**9.9**	37	264
Key(5)	52	1610	38	**6.2**	38	348
Mmgt(4)	12	6.4	8	**0.43**	8	0.70
Q(1)	21	2.1	9	**0.36**	9	0.42
Sentest(400)	413	337	408	**199**	408	471
Speed(1)	7	0.03	4	0.03	4	**0.01**
Tree(100)	100	**18.5**	100	19.8	100	23.2

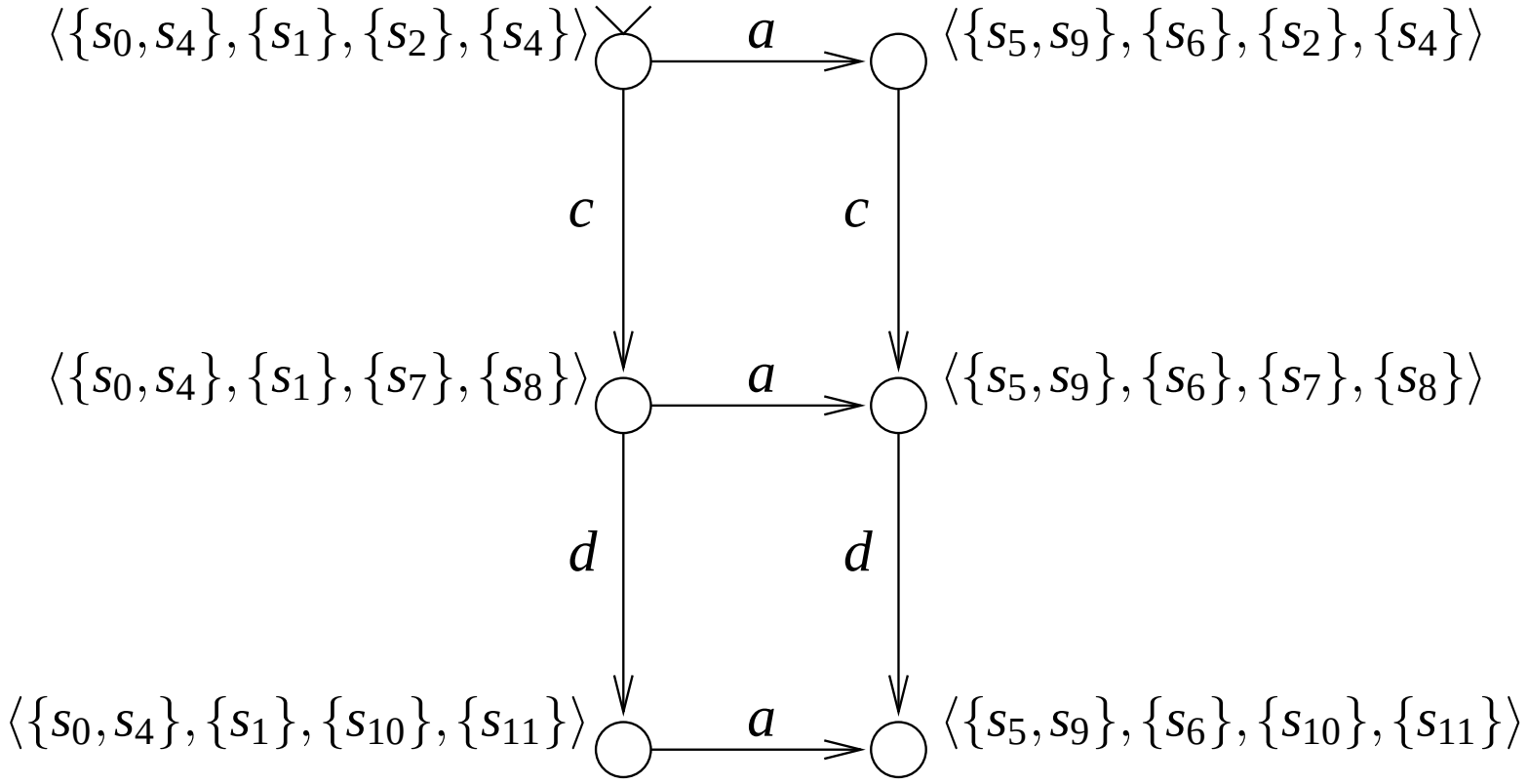
Aalto University

Symbolic Subset Construction

- The FSA subset construction can be used to determinize nondeterministic state machines symbolically inside BMC.
- The tricky part is the correct handling of the τ-moves.
- By doing this, the number of executions through the statespace of the system is further reduced.
- It has also other applications: One can, for example, create a BMC encoding that accepts all words not in the language of L. This has uses, for example, in refinement checking of two products of LTSs.

LTSs: Determinised Interleaving

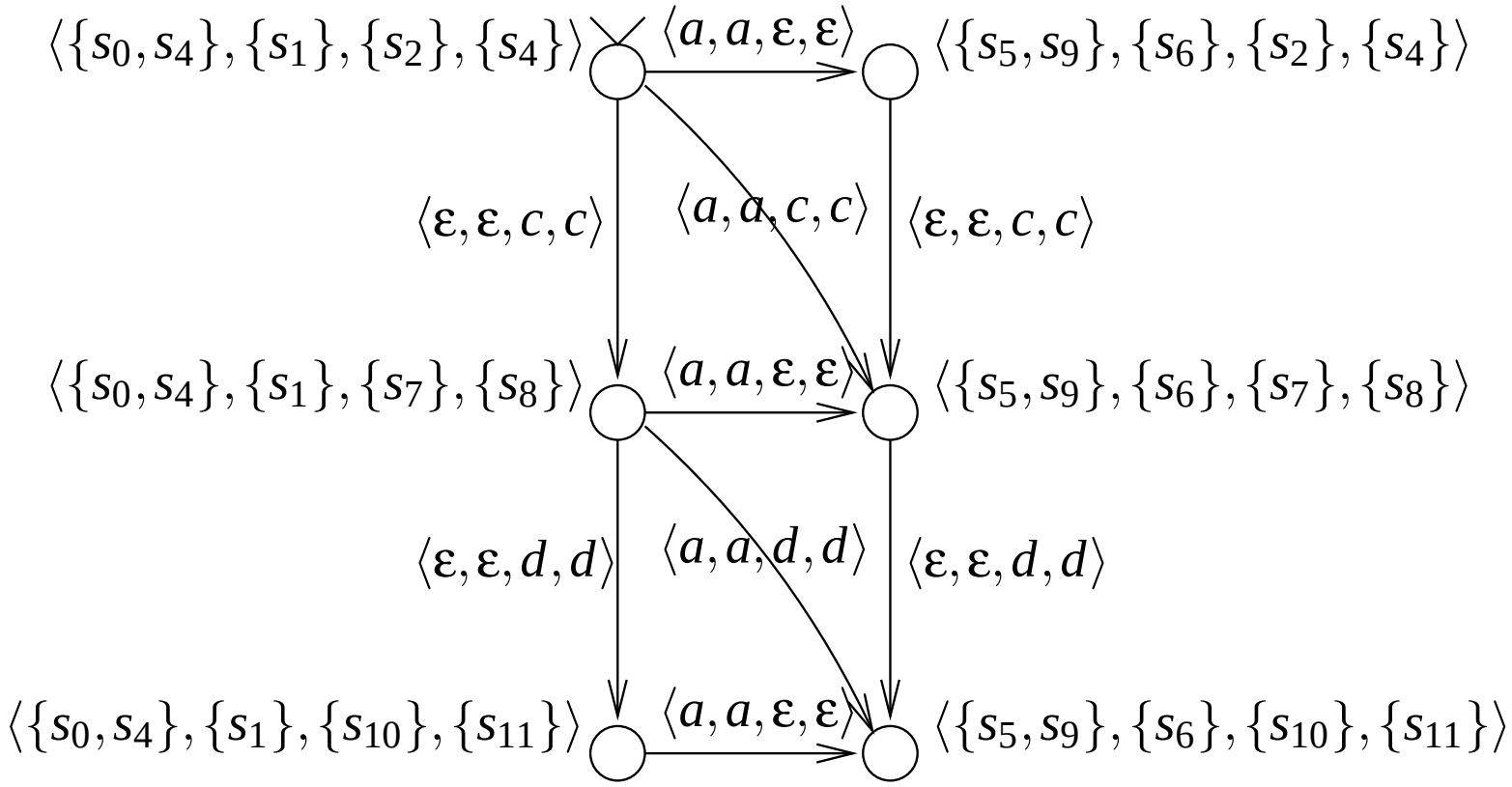
Interleaving combined with determinising each component symbolically during BMC:



Aalto University

LTSs: Determinised Step

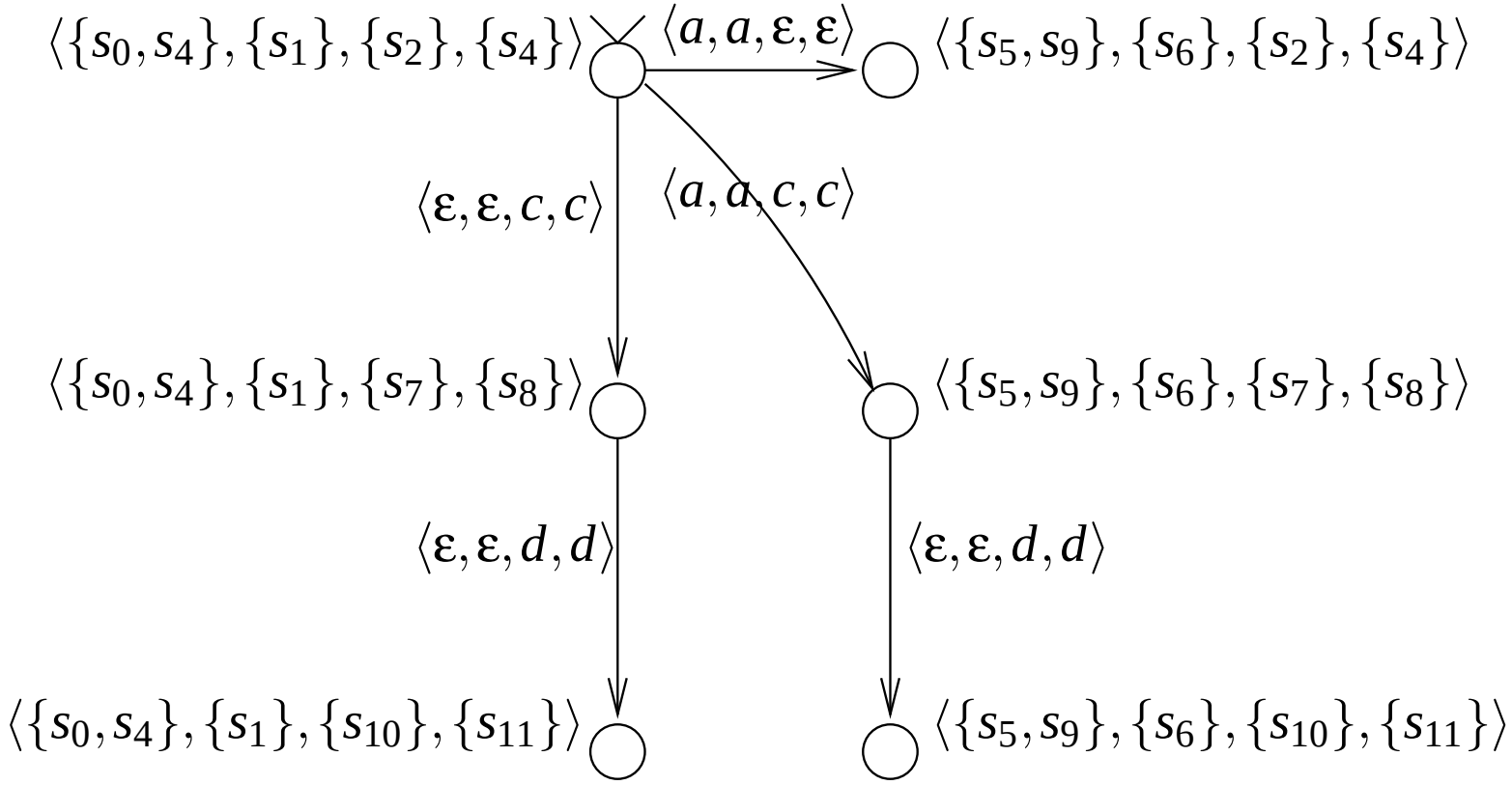
Steps combined with determinising each component symbolically during BMC:



Aalto University

LTSs: Determinised Process

Processes combined with determinising each component symbolically during BMC:



Aalto University

On-the-fly Determinization Results

Table 7.3: Test Results of Interleaving, Step and Process Models (Static Ar
ysis and On-the-fly Determinization)

Problem	Il. k	Il. s	St. k	St. t	Pr. k	Pr. s
Dac(200)	2	0.14	2	0.1	2	**0.07**
Dartes	31	2.9	31	**1.3**	31	2.9
Dp(12)	12	$> 1h$	1	**0.00**	1	**0.00**
Elev(4)	10	2.6	9	1.6	9	**1.2**
Hart.(100)	200	84.3	200	**74.4**	200	79
Key(4)	47	98.4	37	51.7	37	**36.3**
Key(5)	49	450	38	**21.0**	38	36
Mmgt(4)	8	1.7	8	**0.58**	8	0.60
Q(1)	19	0.88	9	0.18	9	**0.16**
Sentest(400)	412	786	408	**56.4**	408	236
Speed(1)	7	0.04	4	0.01	4	**0.00**
Tree(100)	100	**7.4**	100	10.5	100	14.7

Model Checking LTL-X

- One can also do model checking of the temporal logic LTL-X with step semantics.
- LTL-X is the subset of LTL where the next-time operator X has been removed. This restriction of the logic is often done also with other partial order methods.
- For details, see:
  Keijo Heljanko, Ilkka Niemelä:
  Bounded LTL model checking with stable models.
  TPLP 3(4-5): 519-550 (2003), Cambridge University Press.

Steps and AI Planning

- In AI planning papers a step like optimization to decrease the needed bounds was already used.
  Henry A. Kautz, Bart Selman: Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. AAAI/IAAI, Vol. 2 1996: 1194-1201.
- A generalization of step executions, which allows a set of actions S to be fired as a step if at least one interleaving of S is executable is presented in:
  Rintanen, J., Heljanko, K., and Niemelä, I.: Planning as Satisfiability: Parallel Plans and Algorithms for Plan Search Artificial Intelligence 170(12-13):1031-1080.

Other Semantics for BMC

- Other new and efficient non-standard execution semantics for BMC of asynchronous systems have been presented in: Toni Jussila. On bounded model checking of asynchronous systems. Research Report A97, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, October 2005. Doctoral dissertation.
- Another interesting approach is: Shougo Ogata, Tatsuhiro Tsuchiya, Tohru Kikuno: SAT-Based Verification of Safe Petri Nets. ATVA 2004: 79-92, LNCS 3299.

Conclusions of Tutorial part 1

- Bounded model checking (BMC) is an efficient way of implementing *symbolic model checking* complementing other model checking methods.
- It alleviates the state explosion by representing the state space implicitly as a propositional formula.
- It leverages efficient SAT-solver technology.
- The choice between different transition relation encodings has been often overlooked in BMC.
- The performance differences between different transition relation encodings are very significant for asynchronous systems BMC.