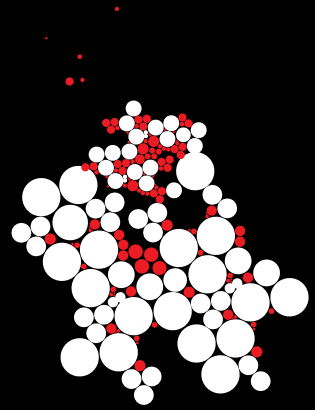
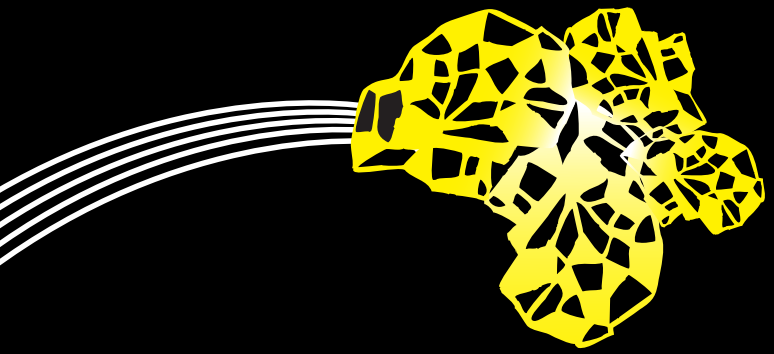
UNIVERSITY OF TWENTE.



Multi-core symbolic bisimulation minimisation

Tom van Dijk & Jaco van de Pol
Formal Methods and Tools
University of Twente, The Netherlands



Multi-core symbolic bisimulation minimisation

Bisimulation minimisation

Given some labeled transition system, compute the smallest equivalent system for some notion of equivalence.

Symbolic methods

Represent state space and transition relation as Boolean functions using binary decision diagrams (BDDs) and multi-terminal BDDs (MTBDDs).

Multi-core programming

Perform the computation in parallel to exploit the power of modern multi-core machines.

Multi-core symbolic bisimulation minimisation

Context

	Explicit-state		Symbolic	
Non-parallel	Paige and Tarjan	1987	Bouali and De Simone	1992
	Groote and Vaandrager	1990	Derisavi	2007
	Van Glabbeek and Weijland	1996	Wimmer, Herbstritt, Hermanns	2007–2010
			Mumme and Ciardo	2013
Parallel	Blom and Orzan	2003	This work	2016
	Kulakowski	2013		

Standing on the shoulders of...

- Signature-based approach by Blom and Orzan
- Signature-based symbolically by Wimmer et al

Multi-core symbolic bisimulation minimisation

Our contributions in the paper

- Use our parallel decision diagram package Sylvan to perform multi-core symbolic bisimulation minimisation.
- Specialized BDD algorithms for partition refinement.
  - `refine` reuses block numbers for stable blocks
  - `inert` restricts the transition relation to inert transitions
- Versatile tool/framework SigRefMC, currently supports strong and branching bisimulation for LTS, CTMC, IMC.
- Benchmarks: up to 95x faster compared to state-of-the-art (before parallel); parallel speedup up to 17x with 48 cores.

CTMC models			Time			Speedups		
Model	States	Blocks	T_w	T_1	T_{48}	Seq.	Par.	Total
p2p-5-6	2^{30}	336	750.29	26.96	2.99	27.83	9.03	251.24
p2p-6-5	2^{30}	266	248.17	9.49	1.21	26.15	7.82	204.47
p2p-7-5	2^{35}	336	2280.76	24.01	2.97	94.99	8.08	767.12

Outline

Bisimulation minimisation

Computing with signature-based partition refinement

Specialized BDD operations
- Sylvan: multi-core decision diagrams
- Assigning block numbers
- Computing inert transitions

Tool support

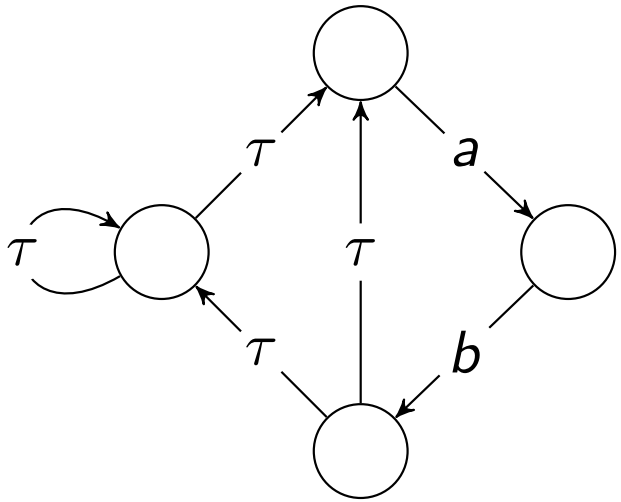
Benchmark results

Conclusions

Labeled transition system

Labeled transition system

- States S
- Transition labels Λ
- Transition relation $\rightarrow : S \times \Lambda \times S$
- Internal transitions are labeled with τ



Bisimulation minimisation

What?

- Compute the smallest LTS with "equivalent behavior".
  (That preserves useful properties)

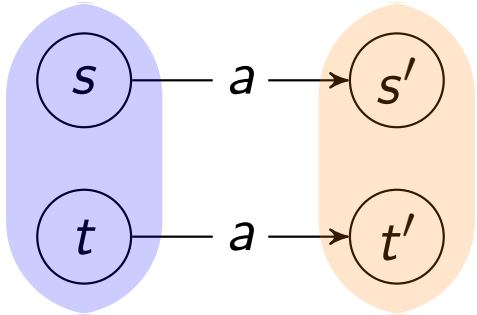
Why?

- State spaces are very large, even for small systems.
- Composition of systems results in exponential growth.
- Often interested in abstractions of the transition system.
- Fewer states: easier to prove properties, better for composition.
  (Especially for explicit-state algorithms)

What is equivalent behavior?

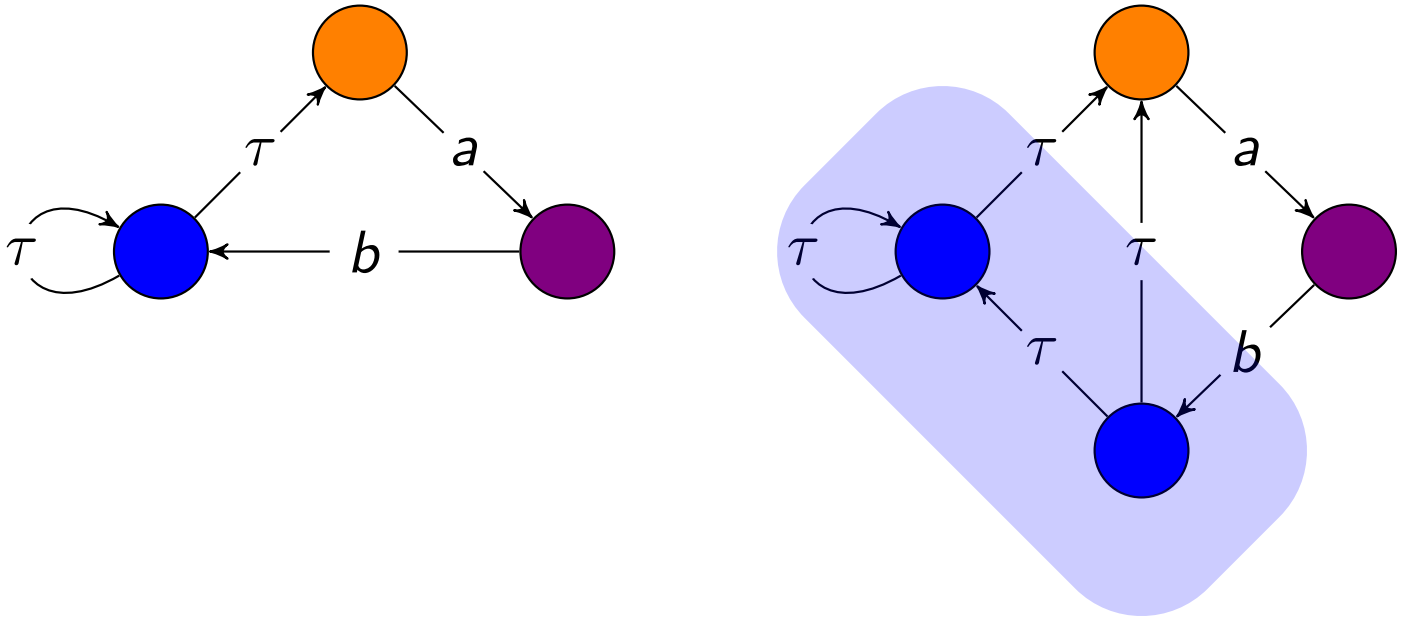
Strong bisimulation

- Two states are equivalent if they have the same transitions.
- If s can do a to s', then t can do a to some t' equivalent to s'.



Strong bisimulation

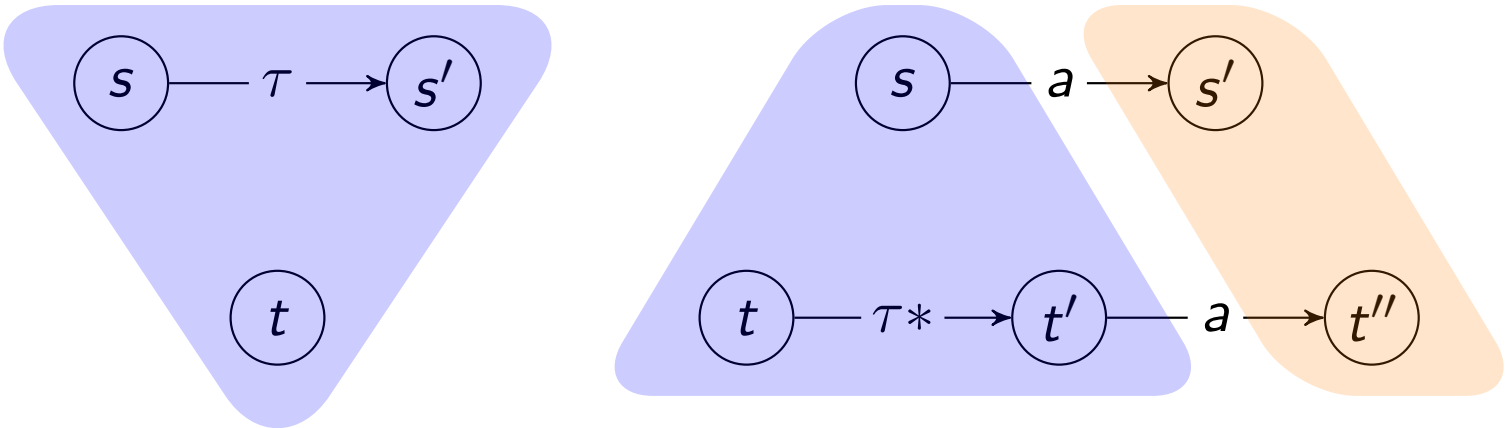
- The following systems are strong bisimilar:
- All ● states can do τ to a ● state and τ to a ● state.



What is equivalent behavior?

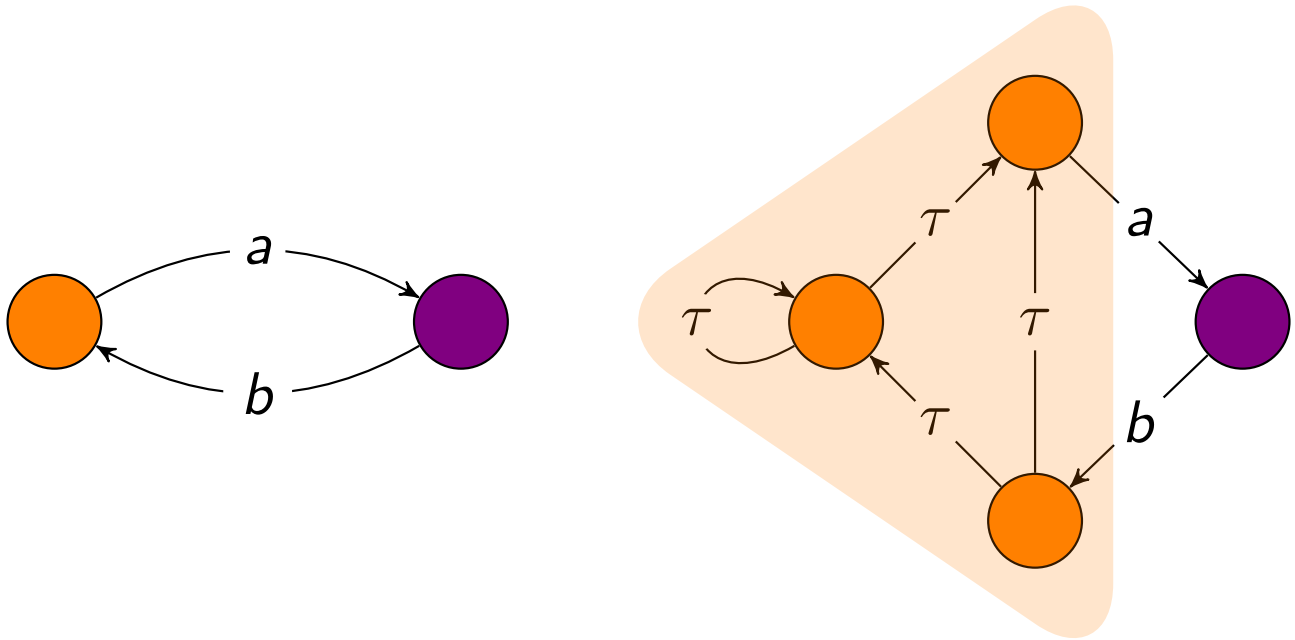
Branching bisimulation

- Two states are equivalent if they have the same transitions, *after taking zero or more τ-transitions to equivalent states*.
- Remove τ self-loops after abstraction.
- If s can do a to s', then t can take "inert" transitions to t' equivalent to t and then a to some t'' equivalent to s'.



Branching bisimulation

- The following systems are branching bisimilar:
- All ● states can all do a to some ● state after internal τ steps.



Outline

Bisimulation minimisation

Computing with signature-based partition refinement

Specialized BDD operations
- Sylvan: multi-core decision diagrams
- Assigning block numbers
- Computing inert transitions

Tool support

Benchmark results

Conclusions

Partitions and signatures

Partition refinement

- Bisimulations partition the state space into blocks.
- Equivalent states have the same *signature* (fingerprint).

Signatures

- Strong bisimulation: all (a, B) such that the state s can do action a to a state in block B.
  $\{(a, B) \mid \exists s' \in B : s \xrightarrow{a} s'\}$.
- Branching bisimulation: all (a, B) such that the state s could do "inert" τ-steps and then do action a to a state in block B.
  $\{(a, B) \mid \exists s' \in B : s \xrightarrow[\mathfrak{B}]{\tau^*} \xrightarrow{a} s' \wedge \neg(a = \tau \wedge s \in B)\}$.

Signature-based partition refinement

Algorithm

1. Start with every state in the same block.
   Partition function $P(s, B)$
2. Calculate the signature of all states.
   Signature function $\sigma(s, a, B)$
   Compute $\sigma(s, a, B)$ from $P(s, B)$ and transitions $T(s, t, a)$
3. Assign block numbers to every unique signature.
   Compute $P(s, B)$ from $\sigma(s, a, B)$
4. Repeat 2 and 3 until no new blocks are generated

Signature-based partition refinement

Strong bisimulation minimisation for LTS

```
1 def minimise(T):
2     P ← block(0)
3     repeat
4         P_prev ← P
5         σ ← and_exists(P, T)
6         P ← refine(σ, P)
7     until P = P_prev
8     return P
```

Outline

Bisimulation minimisation

Computing with signature-based partition refinement

Specialized BDD operations
- Sylvan: multi-core decision diagrams
- Assigning block numbers
- Computing inert transitions

Tool support

Benchmark results

Conclusions

Sylvan: multi-core decision diagrams

- Subject of PhD research on multi-core decision diagrams
- Based on a parallel framework Lace and scalable hash tables
- Automatically computes BDD operations in parallel
- Easy to develop specialized operations

- PDMC 2012: Multi-Core BDD Operations for Symbolic Reachability
  (with Alfons Laarman, Jaco van de Pol)
- TACAS 2015: Sylvan: Multi-core Decision Diagrams
  (with Jaco van de Pol)
- STTT: Sylvan: Multi-core Framework for Decision Diagrams
  (with Jaco van de Pol, invited journal paper, under review)
- SETTA 2015: A Comparative Study of BDD Packages for Probabilistic Symbolic Model Checking
  (with Ernst Moritz Hahn, David N. Jansen, Yong Li, Thomas Neele, Mariëlle Stoelinga, Andrea Turrini, and Lijun Zhang)

Sylvan: multi-core decision diagrams

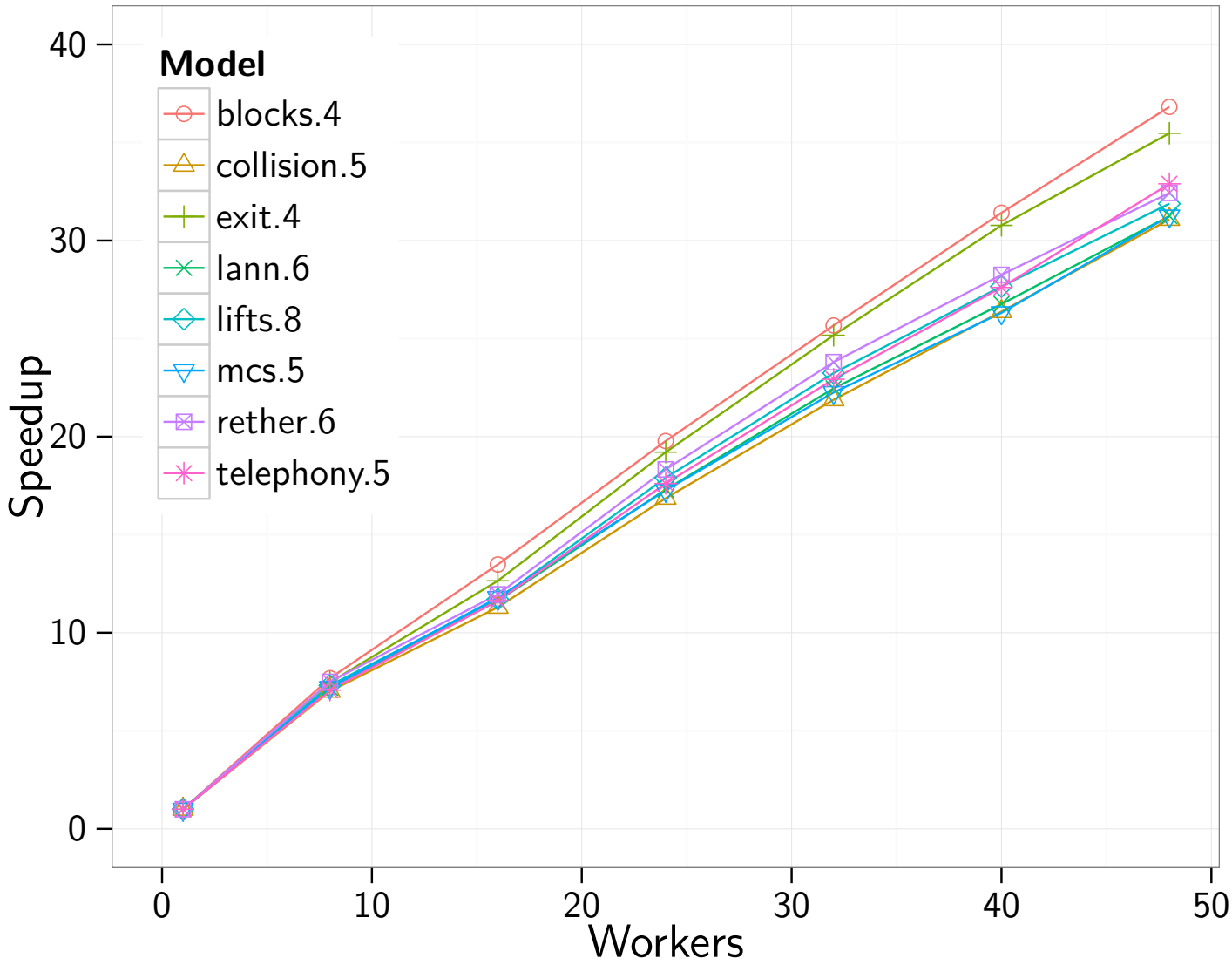
SETTA 2015: A Comparative Study of BDD Packages for Probabilistic Symbolic Model Checking

(with Ernst Moritz Hahn, David N. Jansen, Yong Li, Thomas Neele, Mariëlle Stoelinga, Andrea Turrini, and Lijun Zhang)

- Java-based IscasMC probabilistic model checker
- With multiple cores, Sylvan is faster than the rest
- With only one worker, Sylvan is still pretty fast

backend	time (s)	backend	time (s)
cudd-mtbdd	2837	cudd-bdd	1522
buddy	2156	cacbdd	1433
jdd	2439	beedeedee	2598
sylvan-1	1838	sylvan-7	608

Sylvan


Model
blocks.4
collision.5
exit.4
lann.6
lifts.8
mcs.5
rether.6
telephony.5
Speedup
0
10
20
30
40
Workers
0
10
20
30
40
50

Specialized BDD operations

Motivation for specialized BDD operations

- Off-the-shelf BDD operations (in state-of-the-art SigRef) require multiple steps, e.g. variable renaming.
- Specialized operations perform some computations in 1 step.

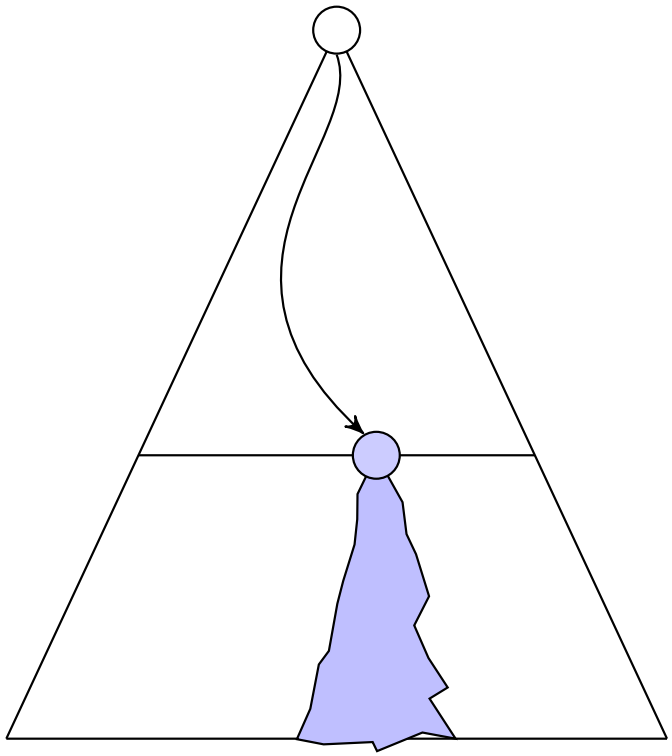
Operations for partition refinement

- `refine`: assigns block numbers, variable renaming, and **reuse previous block assignment** in 1 step.
- `inert`: computes inert τ-transitions in 1 step.

Assigning block numbers

From $\sigma(s, a, B)$ to $P(s, B)$

Function $S \times a \times B$ represents signatures of all states.
Variables of S ordered before a and B.



paths representing
set S of states
(on variables $s_0, \ldots, s_n$)

signatures of the states in S
$(a_0, \ldots, a_A, B_0, \ldots, B_m)$

Assigning block numbers

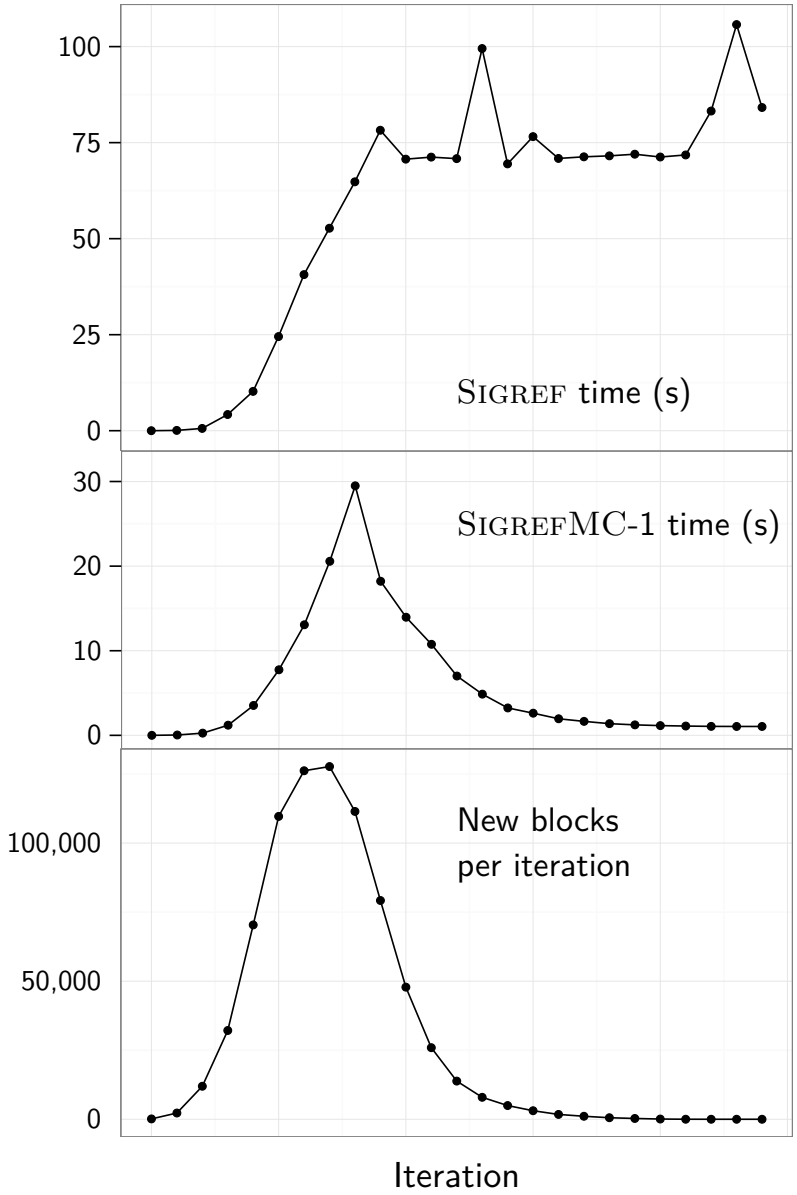
Block number assignment (old Refine)

- Assign a unique number to each signature (C++ dictionary).
- Signatures numbered in the order they are encountered.
- Clear operation caches between iterations.

Block number assignment (new Refine)

- Use previous partition to reuse previous block number.
- New blocks with parallel hash table or parallel skiplist.
- Can use operation cache in next iteration.

Assigning block numbers (kanban model)



Branching bisimulation signature

Inert τ-transitions

Compute transitions $s \xrightarrow{\tau} t$ with Block(s)=Block(t)

$$T(s,t,a) \wedge (a = \tau) \wedge (\exists B\colon P(s,B) \wedge P(t,B))$$

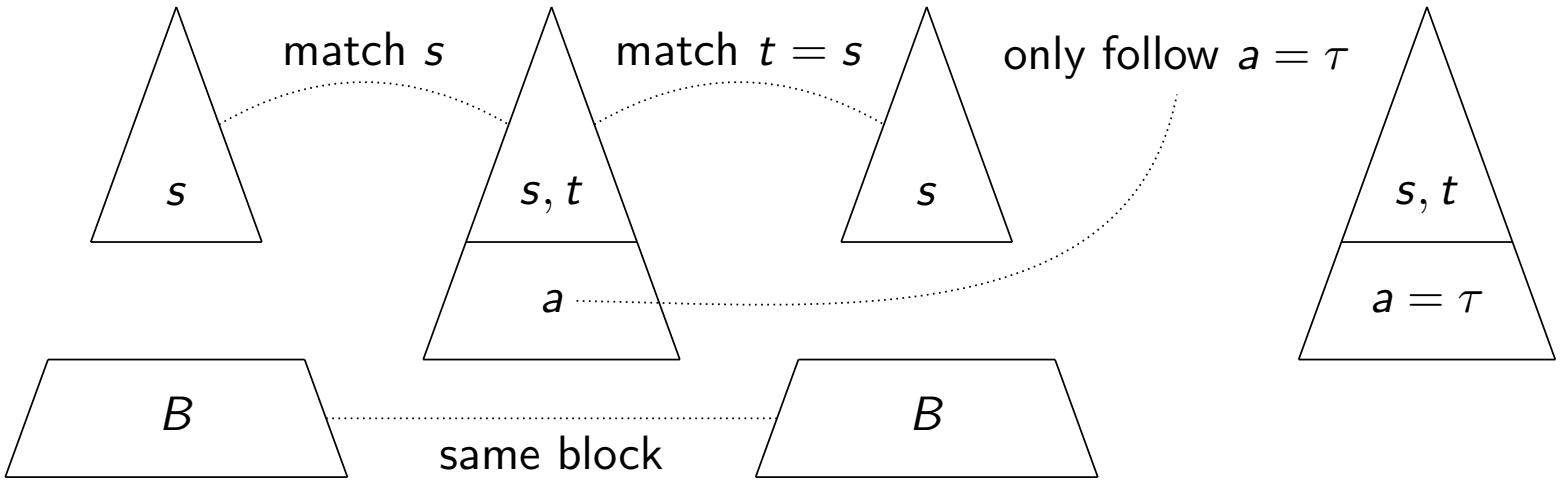
- The intermediate result $\exists B\colon P(s,B) \wedge P(t,B)$ is very large. It actually computes the equivalence relation $\equiv (s,t)$
- Wimmer et al compute one block at a time (with some optimizations).

Branching bisimulation signature

Inert τ-transitions in 1 operation

Compute transitions $s \xrightarrow{\tau} t$ with Block(s)=Block(t)

$$T(s,t,a) \wedge (a = \tau) \wedge (\exists B\colon P(s,B) \wedge P(t,B))$$



P & T & P $\Rightarrow$ inert$_\tau$

Outline

Bisimulation minimisation

Computing with signature-based partition refinement

Specialized BDD operations
- Sylvan: multi-core decision diagrams
- Assigning block numbers
- Computing inert transitions

Tool support

Benchmark results

Conclusions

Tool support

Parallel decision diagram library Sylvan

- http://www.github.com/utwente-fmt/sylvan
- Sylvan now implements a framework for MTBDDs
- Leaf types: integer, double, `mpq_t` (GMP rational numbers)
- Easy to extend with new MTBDD algorithms and types

Multi-core signature refinement tool SigRefMC

- http://www.github.com/utwente-fmt/sigrefmc
- Supports LTS, CTMC, IMC input systems
- Implements signatures for strong and branching bisimulation
- Easy to extend with other signatures and systems

Outline

Bisimulation minimisation

Computing with signature-based partition refinement

Specialized BDD operations
- Sylvan: multi-core decision diagrams
- Assigning block numbers
- Computing inert transitions

Tool support

Benchmark results

Conclusions

Results

Benchmarks

- Compare SigRefMC with state-of-the-art tool SigRef.
- For LTS: Kanban benchmark.
  Models a production environment with four machines each having a parameterizable buffer of workpieces.
- For CTMC: Several benchmarks from PRISM.
- Benchmark machine: 48-core AMD Opteron 6168 1.9GHz.

Results

LTS models (strong)			Time			Speedups		
Model	States	Blocks	T_w	T_1	T_{48}	Seq.	Par.	Total
kanban03	1024240	85356	92.16	10.09	0.88	9.14	11.52	105.29
kanban04	16020316	778485	1410.66	148.15	11.37	9.52	13.03	124.06
kanban05	16772032	5033631	–	1284.86	73.57	–	17.47	–
kanban06	264515056	25293849	–	–	2584.23	–	–	–

LTS models (branching)			Time			Speedups		
Model	States	Blocks	T_w	T_1	T_{48}	Seq.	Par.	Total
kanban04	16020316	2785	8.47	0.52	0.24	16.39	2.11	34.60
kanban05	16772032	7366	34.11	1.48	0.43	22.98	3.47	79.81
kanban06	264515056	17010	118.19	3.87	0.83	30.55	4.65	142.20
kanban07	268430272	35456	387.16	8.83	1.66	43.86	5.31	232.71
kanban08	4224876912	68217	1091.67	17.91	2.98	60.96	6.02	366.72
kanban09	4293193072	123070	3186.48	34.23	5.51	93.10	6.21	578.59

Results

CTMC models			Time			Speedups		
Model	States	Blocks	T_w	T_1	T_{48}	Seq.	Par.	Total
cycling-4	431101	282943	220.23	26.72	2.60	8.24	10.29	84.84
cycling-5	2326666	1424914	1249.23	170.28	19.42	7.34	8.77	64.34
fgf	80616	38639	71.62	8.86	0.88	8.08	10.04	81.20
p2p-5-6	2^{30}	336	750.29	26.96	2.99	27.83	9.03	251.24
p2p-6-5	2^{30}	266	248.17	9.49	1.21	26.15	7.82	204.47
p2p-7-5	2^{35}	336	2280.76	24.01	2.97	94.99	8.08	767.12
polling-16	1572864	98304	792.82	118.50	10.18	6.69	11.64	77.85
polling-17	3342336	196608	1739.01	303.65	22.58	5.73	13.45	77.03
polling-18	7077888	393216	–	705.22	49.81	–	14.16	–
robot-020	31160	30780	28.15	3.21	0.60	8.78	5.36	47.04
robot-025	61200	60600	78.48	6.78	0.95	11.58	7.11	82.39
robot-030	106140	105270	174.30	12.26	1.47	14.21	8.33	118.44

Outline

Bisimulation minimisation

Computing with signature-based partition refinement

Specialized BDD operations
- Sylvan: multi-core decision diagrams
- Assigning block numbers
- Computing inert transitions

Tool support

Benchmark results

Conclusions

Conclusions

Contributions for symbolic signature refinement

1. Signature refinement using partial signatures (see paper)
2. Improved Refine operation that reuses block numbers
3. Compute inert steps in 1 specialized BDD operation
4. Multi-core implementation in tool SigRefMC

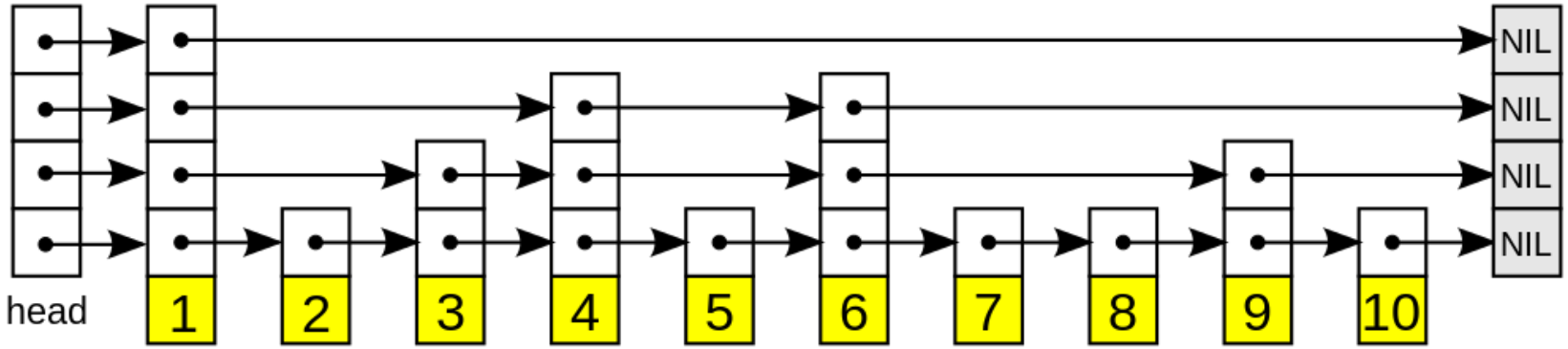
Ideas for future research

- Investigate more parallelism and different benchmarks
- Bisimulation minimisation for MDPs and other types
- Smarter way to get a good symbolic result
  Currently looks like a random graph
- Integrate in toolchains, e.g. with LTSmin toolset

Backup slides

Backup slides

Skiplist



Lockless insertion

- Use a short-lived `cas`-lock on the "next edge" of the immediate predecessor.
  - Only blocks threads that also want to insert there.
- Use global counter to get next block number and write data.
- Update "next edge" on lowest level and release lock.
- Update "next edges" on higher levels using `cas`.