Efficient Model Checking of Safety Properties

Timo Latvala

timo.latvala@hut.fi

Laboratory for Theoretical Computer Science
Helsinki University of Technology
Finland

Introduction

- Safety properties $\approx$ "properties with finite counterexamples".
- Uses:
  - Testing
  - Program monitoring
  - *Model Checking*
- Focus: LTL using the automata theoretic approach.
- Main problem: translating LTL formulas to finite automata.

Why safety properties?

Treating safety properties as a special case has certain benefits.

- Safety properties an “important” subset.

Why safety properties?

Treating safety properties as a special case has certain benefits.

- Safety properties an “important” subset.
- Explicit state model checking algorithms are somewhat simpler.

Why safety properties?

Treating safety properties as a special case has certain benefits.

- Safety properties an “important” subset.
- Explicit state model checking algorithms are somewhat simpler.
- BDD-based algorithms are faster (linear vs quadratic).

Why safety properties?

Treating safety properties as a special case has certain benefits.

- Safety properties an “important” subset.
- Explicit state model checking algorithms are somewhat simpler.
- BDD-based algorithms are faster (linear vs quadratic).
- For methods such as Petri net unfoldings, safety is much simpler.

Challenges

Treating safety as a special case poses some challenges:

- Deciding if an LTL formula is a safety formula is PSPACE-complete.

Challenges

Treating safety as a special case poses some challenges:

- Deciding if an LTL formula is a safety formula is PSPACE-complete.
- Translating a safety LTL formula to finite automaton doubly exponential.

Challenges

Treating safety as a special case poses some challenges:

- Deciding if an LTL formula is a safety formula is PSPACE-complete.
- Translating a safety LTL formula to finite automaton doubly exponential.
- Non-pathological formulas have a singly exponential translation to finite automata.

Challenges

Treating safety as a special case poses some challenges:

- Deciding if an LTL formula is a safety formula is PSPACE-complete.
- Translating a safety LTL formula to finite automaton doubly exponential.
- Non-pathological formulas have a singly exponential translation to finite automata.
- Deciding if a formula is pathological is PSPACE-complete.

Contributions

- A new translation algorithm based on an algorithm by Kupferman and Vardi.

Contributions

- A new translation algorithm based on an algorithm by Kupferman and Vardi.
- Extensive experimental testing of the implementation.

Contributions

- A new translation algorithm based on an algorithm by Kupferman and Vardi.
- Extensive experimental testing of the implementation.
- First(?) implementation of a algorithm checking if a formula is pathologic.

Contributions

- A new translation algorithm based on an algorithm by Kupferman and Vardi.
- Extensive experimental testing of the implementation.
- First(?) implementation of a algorithm checking if a formula is pathologic.
- The tool, *scheck*, can be used with Spin.

Related Work

- Kupferman and Vardi: Algorithms and complexity results.

Related Work

- Kupferman and Vardi: Algorithms and complexity results.
- Geilen: Forward version of KV-algorithm.

Related Work

- Kupferman and Vardi: Algorithms and complexity results.
- Geilen: Forward version of KV-algorithm.
- Berard et al: history variables methods for past TL

Related Work

- Kupferman and Vardi: Algorithms and complexity results.
- Geilen: Forward version of KV-algorithm.
- Berard et al: history variables methods for past TL
- Havelund and Rosu: model checking past TL for finite executions.

Translation Algorithm

- The algorithm creates the finite automaton backwards.

Translation Algorithm

- The algorithm creates the finite automaton backwards.
- We start from an empty set of requirements and analyse the satisfaction of subformulas.

Translation Algorithm

- The algorithm creates the finite automaton backwards.
- We start from an empty set of requirements and analyse the satisfaction of subformulas.
- We only add states for temporal operators (exception: X).

Translation Algorithm

- The algorithm creates the finite automaton backwards.
- We start from an empty set of requirements and analyse the satisfaction of subformulas.
- We only add states for temporal operators (exception: X).
- Resulting automaton accepts all *informative* prefixes.

Checking Pathologic Safety

- Construct $\mathcal{A}_{\neg\psi}$.

Checking Pathologic Safety

- Construct $\mathcal{A}_{\neg\psi}$.
- Construct *deterministic* finite automaton $\mathcal{B}_{\neg\psi}$.

Checking Pathologic Safety

- Construct $\mathcal{A}_{\neg\psi}$.
- Construct *deterministic* finite automaton $\mathcal{B}_{\neg\psi}$.
- Interpret $\mathcal{B}_{\neg\psi}$ as a Büchi automaton and complement it.

Checking Pathologic Safety

- Construct $\mathcal{A}_{\neg\psi}$.
- Construct *deterministic* finite automaton $\mathcal{B}_{\neg\psi}$.
- Interpret $\mathcal{B}_{\neg\psi}$ as a Büchi automaton and complement it.
- If $\mathcal{L}(\mathcal{A}_{\neg\psi} \times \bar{\mathcal{B}}_{\neg\psi}) \neq \emptyset$ then ψ is pathologic.

Implementation

- The implementation uses BDDs to manage sets.

Implementation

- The implementation uses BDDs to manage sets.
- Produces deterministic or non-deterministic automata.

Implementation

- The implementation uses BDDs to manage sets.
- Produces deterministic or non-deterministic automata.
- Can be connected to Spin.

Implementation

- The implementation uses BDDs to manage sets.
- Produces deterministic or non-deterministic automata.
- Can be connected to Spin.
- Freely available licensed under the GNU GPL.

Experiments

- Randomly generated syntactically safe formulas

Experiments

- Randomly generated syntactically safe formulas
- Randomly generated formulas.

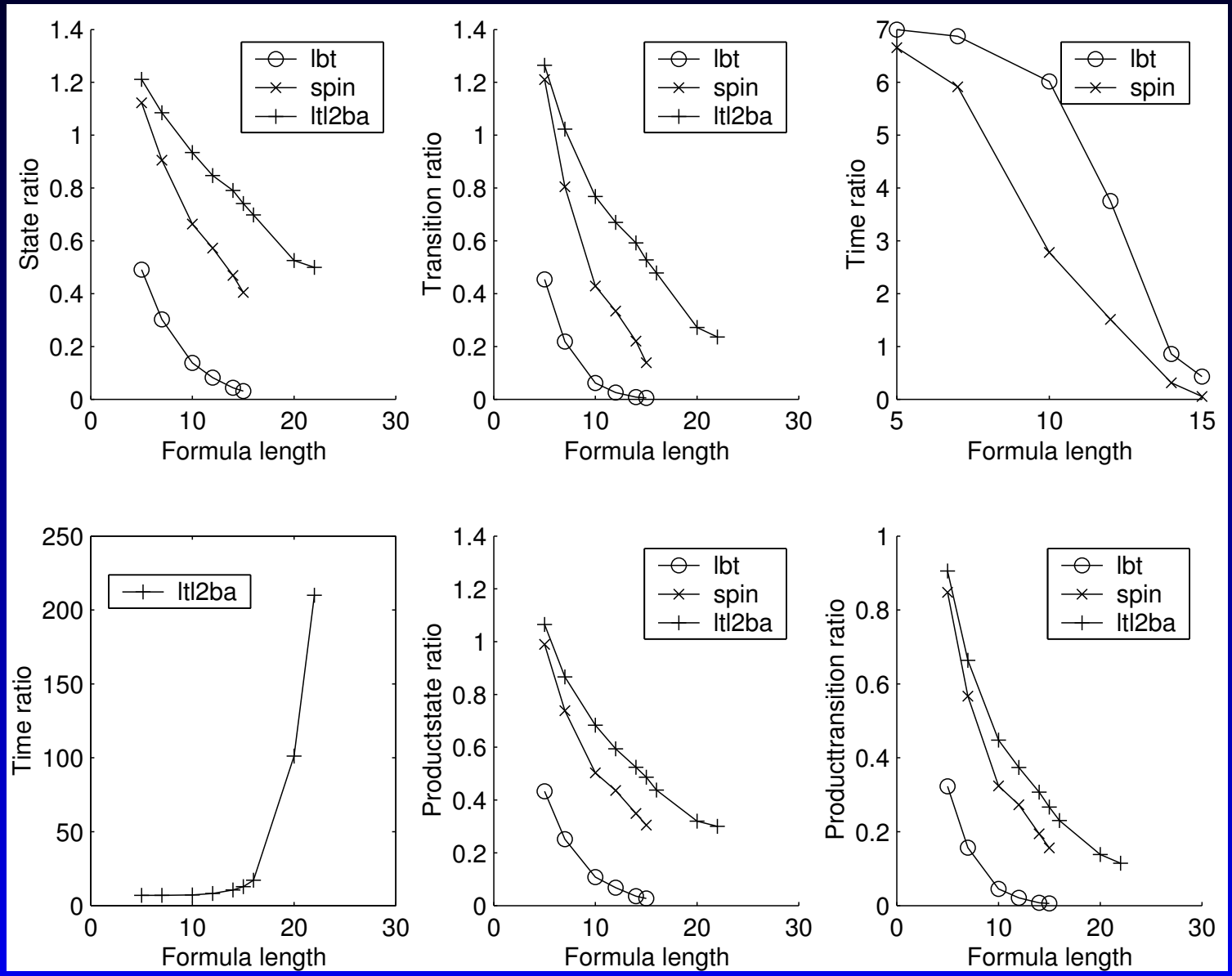
Experiments

- Randomly generated syntactically safe formulas
- Randomly generated formulas.
- Safety formulas from the specification pattern system.

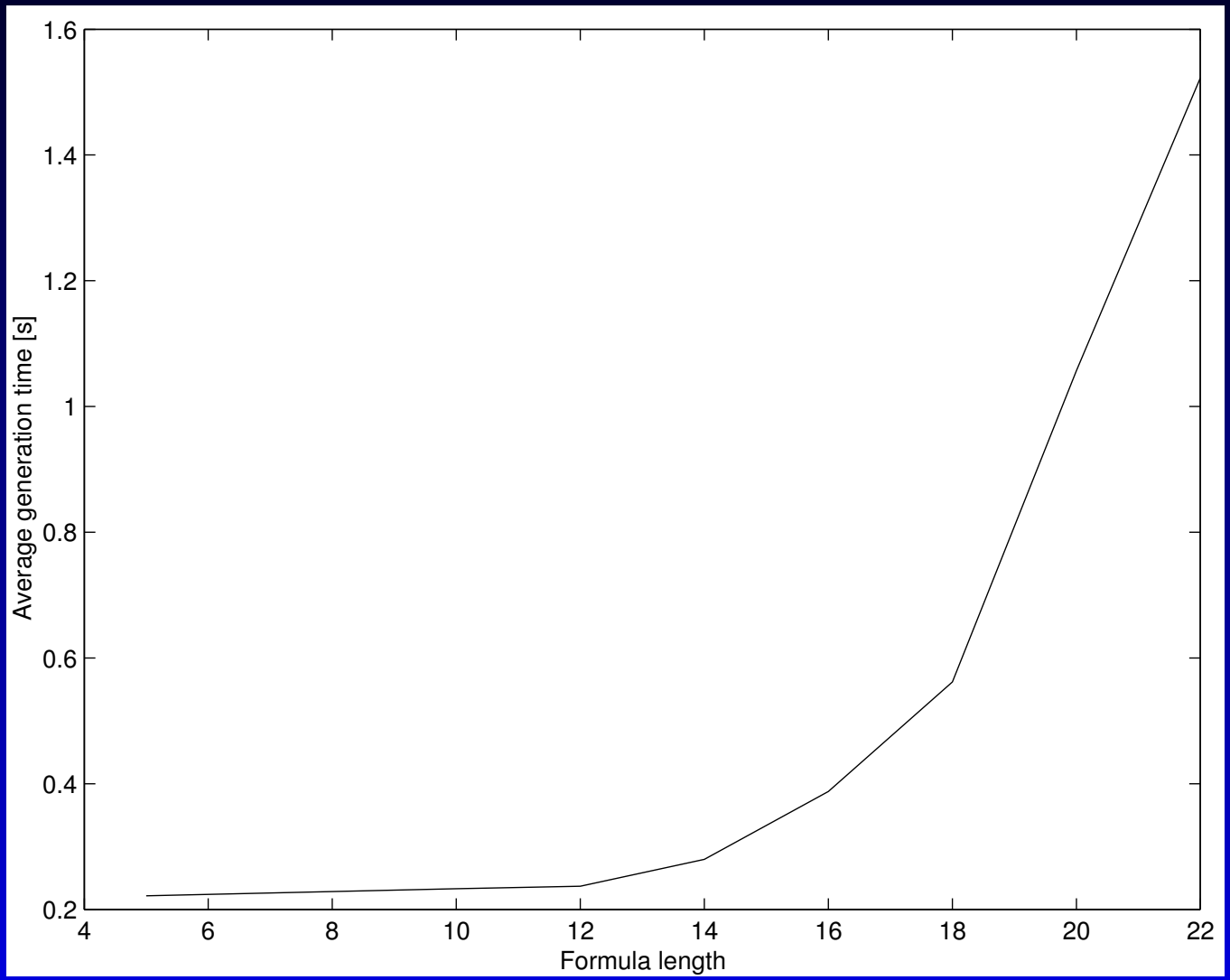
Experiments

- Randomly generated syntactically safe formulas
- Randomly generated formulas.
- Safety formulas from the specification pattern system.
- Model checking tests with Spin.

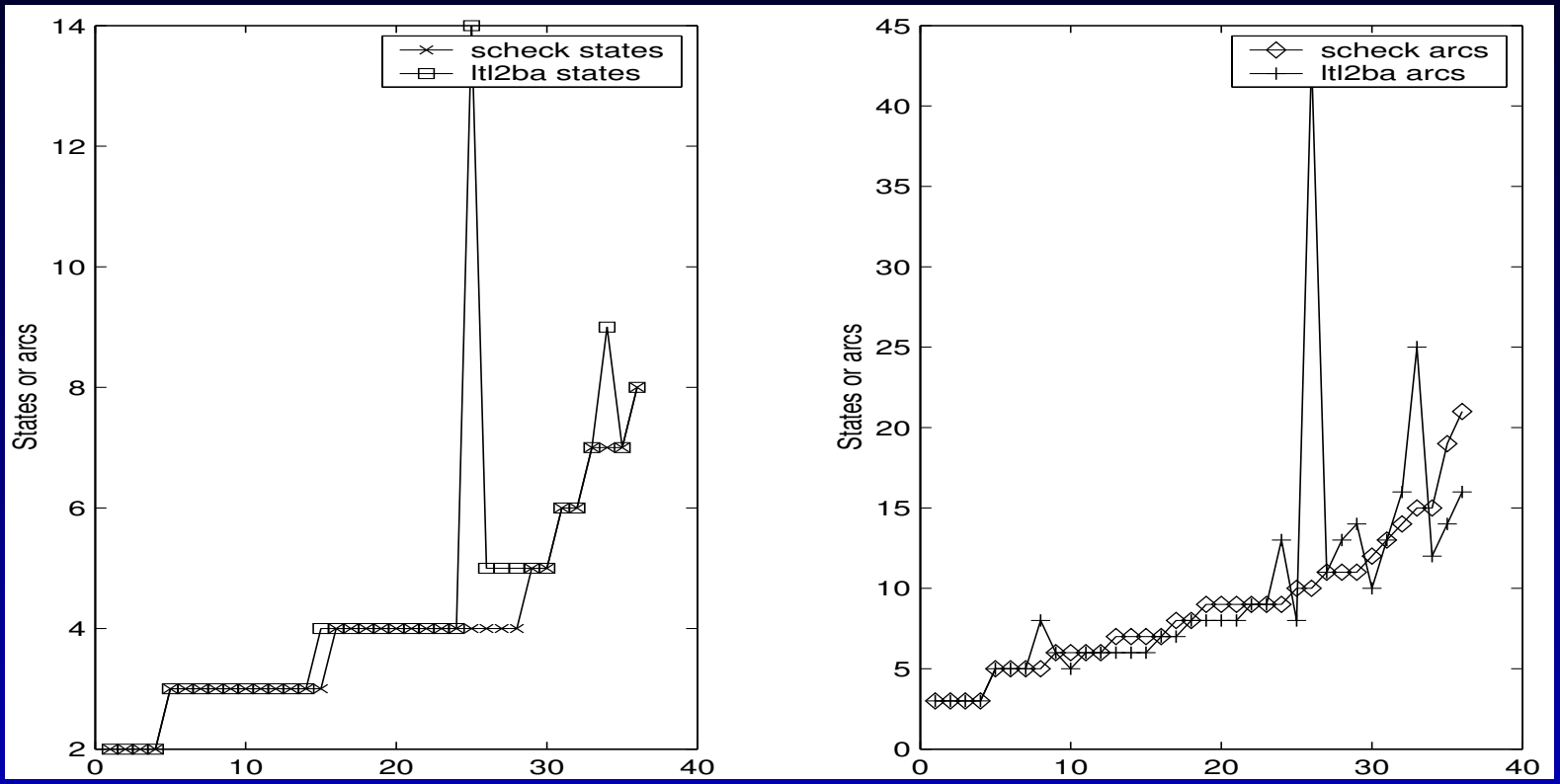
Syntactically Safe Formulas



General Formulas



Specification Pattern Formulas



	states	arcs	time [s]	product states	product arcs
ltl2ba	160	348	0.5	3037	15406
lbt	1915	31821	1.2	25134	763203
scheck	144	316	2.1	2481	9806

Practical Models

model	*scheck*			*spin*		
	states	arcs	t [s]	states	arcs	t [s]
peterson(3)	17476	32343	0.06	21792	45870	0.09
peterson(4)	3254110	709846	20.8	4216030	10315000	37.3
sliding(1,1)	130799	407238	0.9	258456	890026	2.2
sliding(1,2)	518050	1670120	3.9	1027130	3604660	9.8
sliding(2,1)	5447700	18271400	534.7	10794100	39649800	1097.4
erathostenes(50,1)	522	522	0.03	522	522	0.03
erathostenes(60,2)	324	324	0.02	357958	647081	4.0
erathostenes(70,3)	522	522	0.04	2047030	4407400	48.5
erathostenes(80,4)	789	789	0.04	-	-	-
erathostenes(80,5)	847	847	0.04	-	-	-
iprot	7095180	20595400	377.0	16011900	46288600	1006.2
giop	146646	215640	1.8	255105	524493	4.8

Practical Models

model	*scheck*			*ltl2ba*		
	states	arcs	t [s]	states	arcs	t [s]
peterson(3)	17476	32343	0.06	21792	45870	0.09
peterson(4)	3254110	709846	20.8	4216030	10315000	37.5
sliding(1,1)	130799	407238	0.09	258432	890386	2.2
sliding(1,2)	518050	1670120	3.9	1027120	3604410	9.8
sliding(2,1)	5447700	18271400	534.7	10794000	39645700	1097.6
erathostenes(50,1)	522	522	0.03	678	678	0.03
erathostenes(60,2)	324	324	0.02	794322	1319710	8.4
erathostenes(70,3)	522	522	0.04	3110700	6474410	76.6
erathostenes(80,4)	789	789	0.04	-	-	-
erathostenes(80,5)	847	847	0.04	-	-	-
iprot	7095180	20595400	377.0	16011900	46288600	1003.7
giop	146646	215640	1.8	255105	524493	4.6

Conclusions

- *scheck* produces smaller automata is most cases.

Conclusions

- *scheck* produces smaller automata is most cases.
- Especially when debugging safety properties, the gain can be significant

Conclusions

- *scheck* produces smaller automata is most cases.
- Especially when debugging safety properties, the gain can be significant
- A model checker can gain by analysing the formula.

Conclusions

- *scheck* produces smaller automata is most cases.
- Especially when debugging safety properties, the gain can be significant
- A model checker can gain by analysing the formula.
- Using BDDs probably a bad design choice.

Conclusions

- *scheck* produces smaller automata is most cases.
- Especially when debugging safety properties, the gain can be significant
- A model checker can gain by analysing the formula.
- Using BDDs probably a bad design choice.
- *scheck* is available from `www.tcs.hut.fi/~timo/scheck`.