Chiefly Symmetric: Results on the Scalability of Probabilistic Model Checking for Operating-System Code

Christel Baier[1], Marcus Daum[1], Benjamin Engel[2], Hermann Härtig[2], Joachim Klein[1], Sascha Klüppelholz[1], Steffen Märcker[1], Hendrik Tews[2], Marcus Völp[2]

[1] Institute for Theoretical Computer Science

[2] Operating-Systems Group

Technische Universität Dresden, Germany

Systems Software Verification Conference, November 30, 2012

Motivation

Many systems are not safety critical

- Soft Real-Time systems, Hard real-time systems with safe failback
  - entertainment in aeroplanes and cars
- meet deadlines with high probability, users will tolerate rare misses
- quantitative probabilistic properties are relevant
  - Video is played without interrupts with a probability of 99%

QuaOS Project — Quantitative Analysis of Operating Systems

- determine quantitative probabilistic properties of OS code
  - probability for serving the interrupt within 200 ns
  - the time quantile in which 99.9% of interrupts are served
- use probabilistic model checking
- determine soft real-time guarantees
- predict properties which cannot be measured (yet)
  - need to reproduce measurements exactly

Previous Work

FMICS 2012 : Waiting for locks: How long does it usually take?

- model a Test-and-test-and-set lock in PRISM
- Properties checked (steady state – i.e., in an infinitely long run)
  - probability to acquire the lock without waiting
  - expected waiting time
  - 95% quantile of the waiting time
- PRISM reproduced results from measurements almost exactly
- bounded scalability
  - 4 processes
  - distribution for critical region only 1 sampling point
  - distribution for non-critical region with 2–4 sampling points

This Talk

Present model specific symmetry reduction

- scales up to 10,000 processes
- using MRMC and a custom program for generating the DTMC model
- scalability relies on the total saturation of the lock
  - all but 9 processes are spinning
- simple spin lock permits to study the benefits of symmetry reduction
- (results are not directly relevant for spin locks but for other high contention locks)

Outline

- Introduction
- A test-and-test-and-set Lock and its DTMC model
- Symmetry Reduction
- Results for the reduced model
- Conclusion

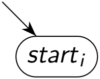
Test-And-Test-And-Set Lock

```
volatile bool occupied = false;

volatile void lock(){
  while (atomic_swap(occupied, true)){
      while (occupied){/* spin loop */}
  }
}
void unlock(){
  occupied = false
}
```

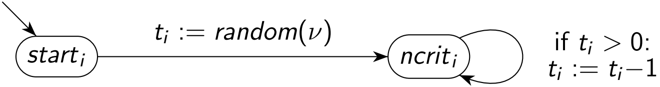
- model *n* processes that compete for the lock
- model lock as separate process
- compare results with measurements

Process i : DTMC model



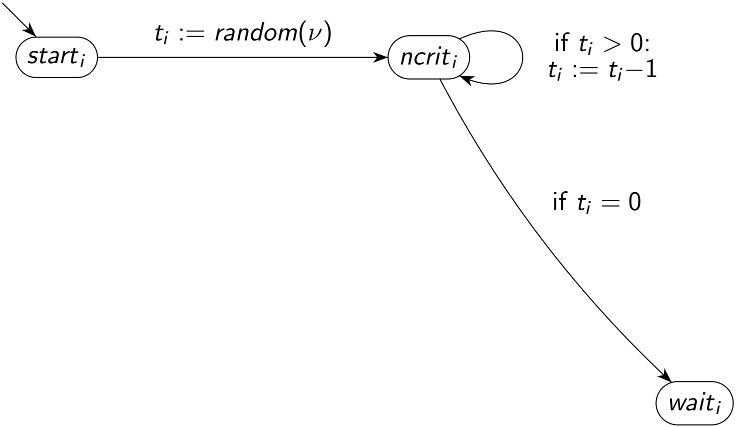
Distributions: ν non-critical region

Process i : DTMC model



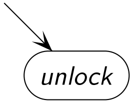
Distributions: ν non-critical region

Process i : DTMC model

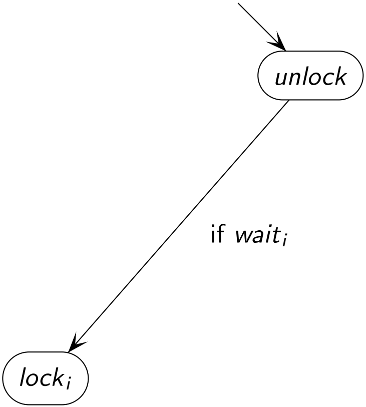


Distributions: ν non-critical region

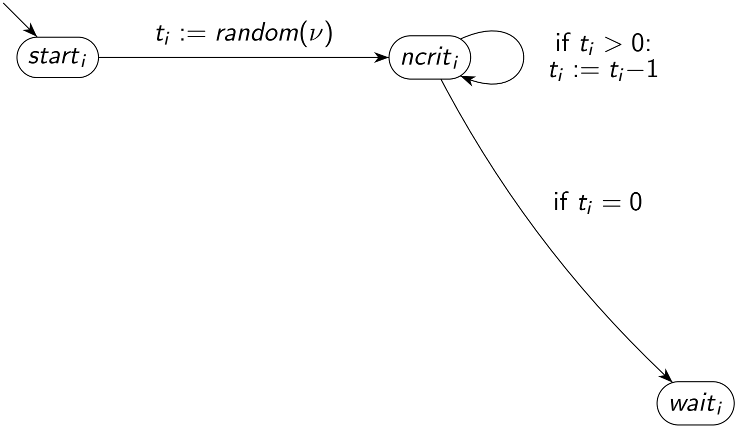
The lock: DTMC model



The lock: DTMC model



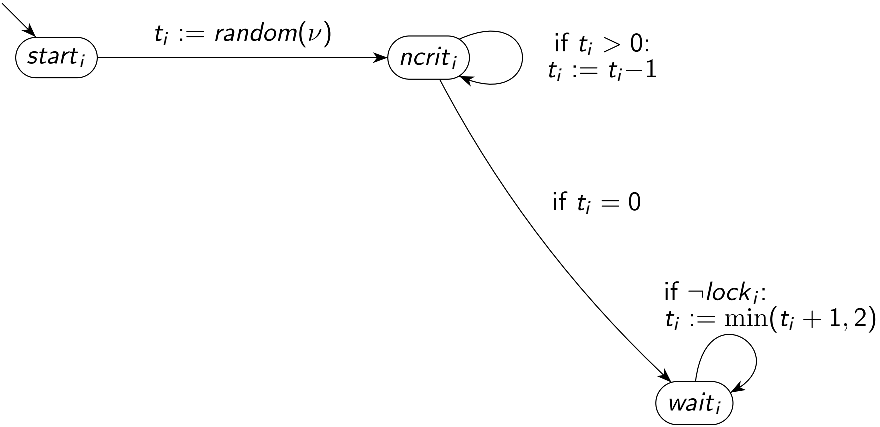
Process i: DTMC Model



Distributions:

ν	non-critical region
γ	critical region
γ_1	critical region (with spinning)

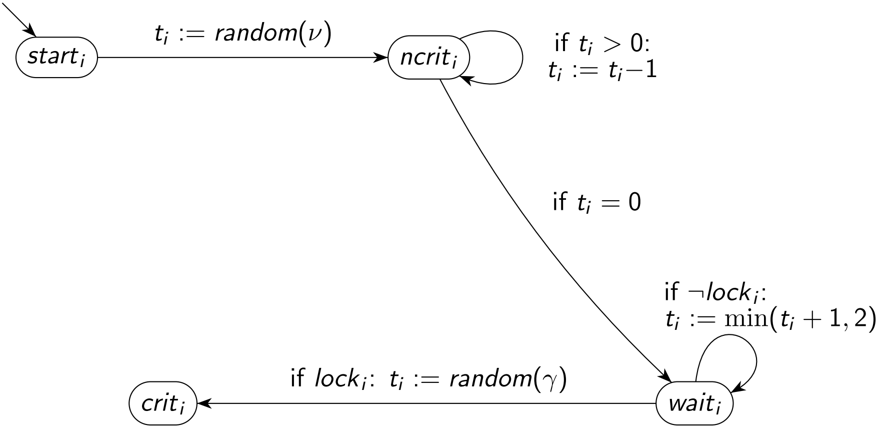
Process i: DTMC Model



Distributions:

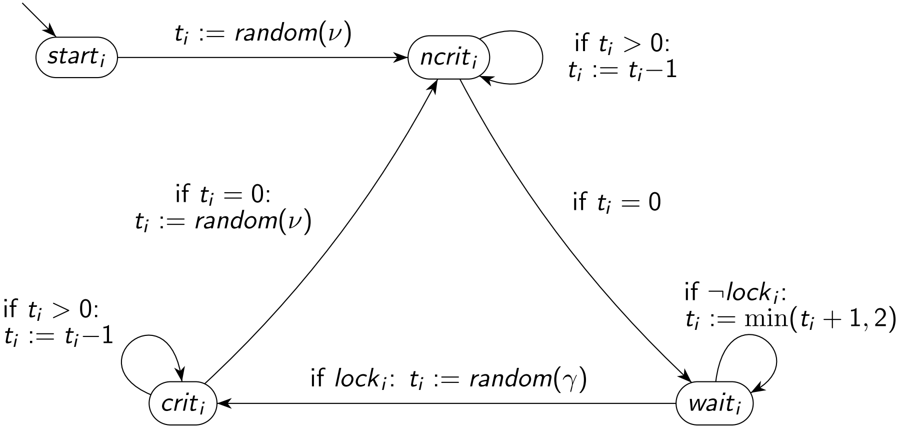
ν	non-critical region
γ	critical region
γ_1	critical region (with spinning)

Process i: DTMC Model



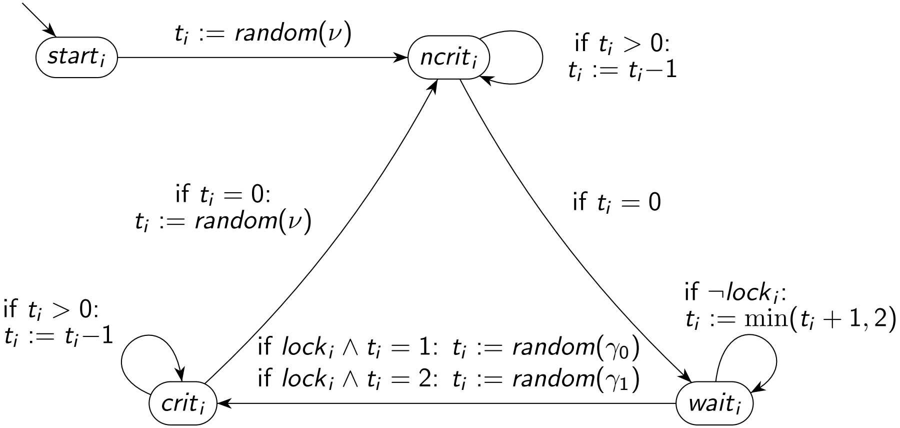
Distributions:		
	ν	non-critical region
	γ	critical region
	γ_1	critical region (with spinning)

Process i: DTMC Model



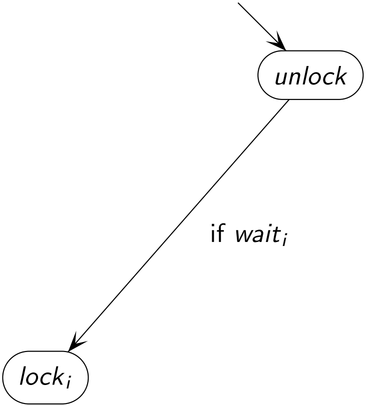
Distributions:	ν	non-critical region
	γ	critical region
	γ_1	critical region (with spinning)

Process i: DTMC Model



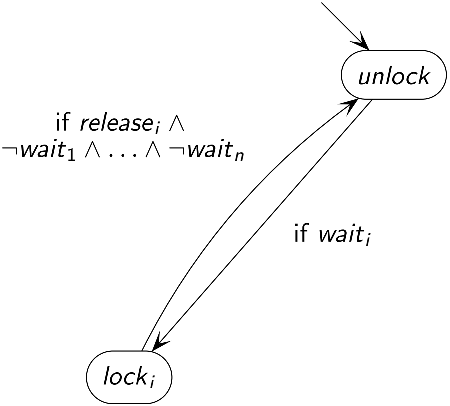
Distributions:		
	ν	non-critical region
	γ_0	critical region (without spinning)
	γ_1	critical region (with spinning)

The lock: DTMC Model



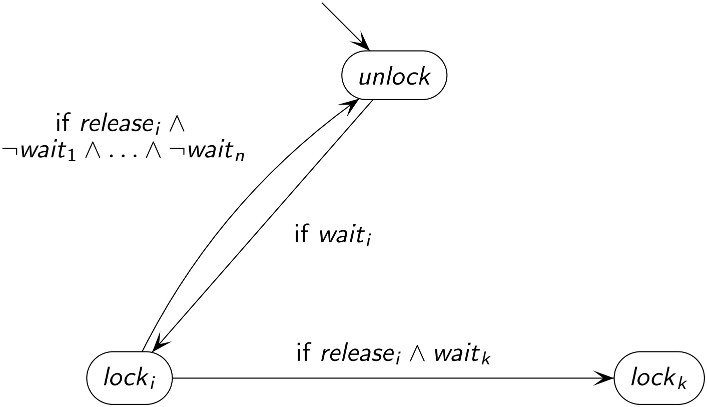
perform uniform probabilistic choice for selecting next lock owner

The lock: DTMC Model



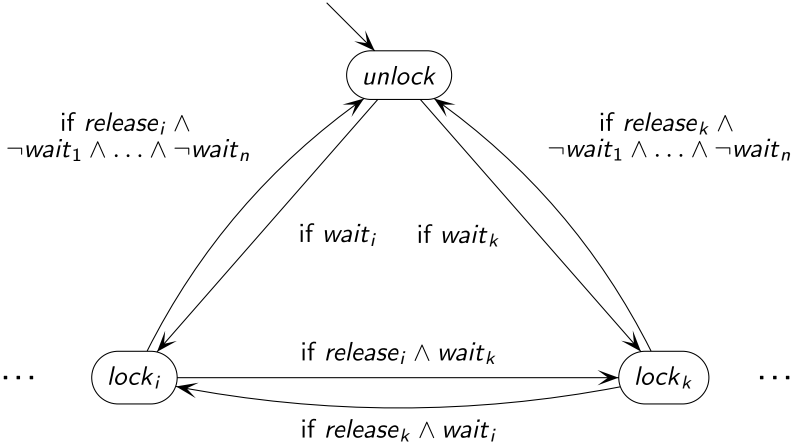
perform uniform probabilistic choice for selecting next lock owner

The lock: DTMC Model


unlock
if $release_i \wedge$
$\neg wait_1 \wedge \ldots \wedge \neg wait_n$
if $wait_i$
if $release_i \wedge wait_k$
$lock_i$
$lock_k$

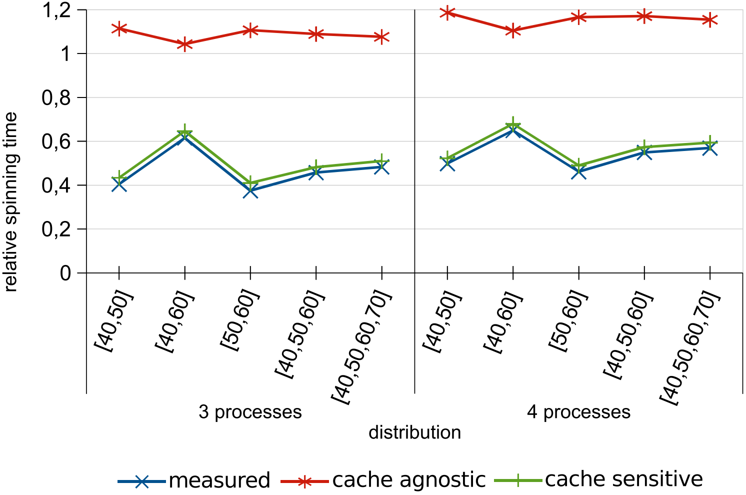
perform uniform probabilistic choice for selecting next lock owner

The lock: DTMC Model

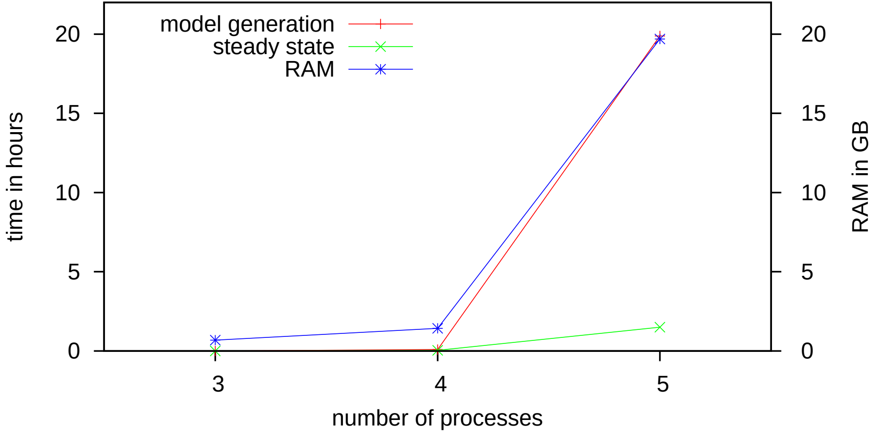

unlock
if $release_i \wedge$
$\neg wait_1 \wedge \ldots \wedge \neg wait_n$
if $release_k \wedge$
$\neg wait_1 \wedge \ldots \wedge \neg wait_n$
if $wait_i$
if $wait_k$
if $release_i \wedge wait_k$
$\cdots$
$lock_i$
$lock_k$
$\cdots$
if $release_k \wedge wait_i$

perform uniform probabilistic choice for selecting next lock owner

Results presented at FMICS: expected spinning time



Scalability for PRISM, Distribution [40,50]



number of states:

3 proc.	4,082,808
4 proc.	198,808,720

Outline

Introduction

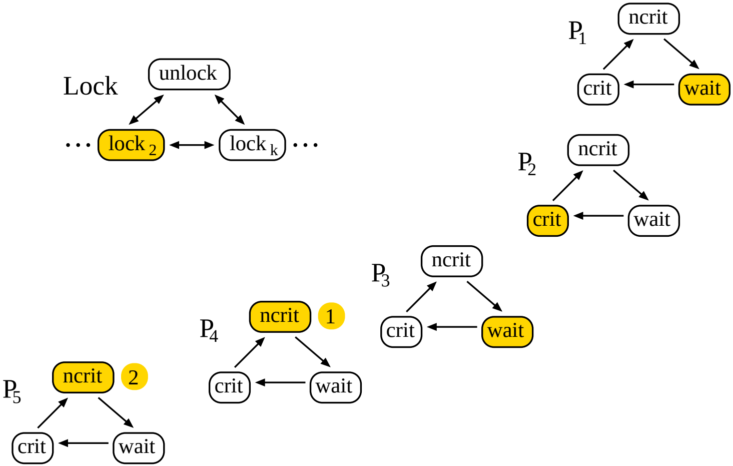
A test-and-test-and-set Lock and its DTMC model

Symmetry Reduction

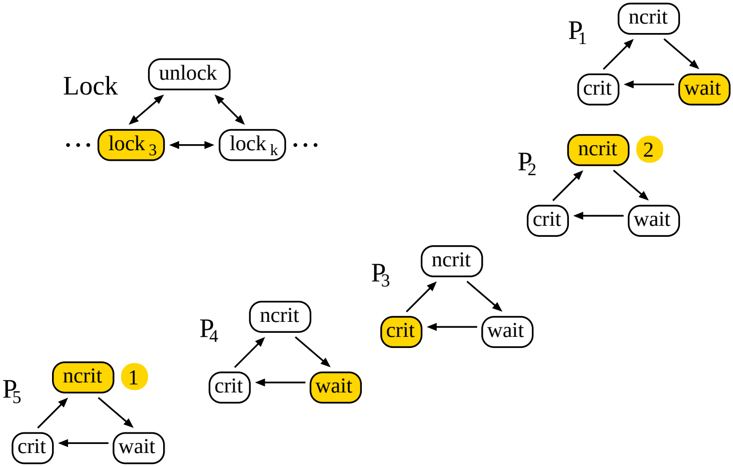
Results for the reduced model

Conclusion

Symmetry in the model



Symmetry in the model

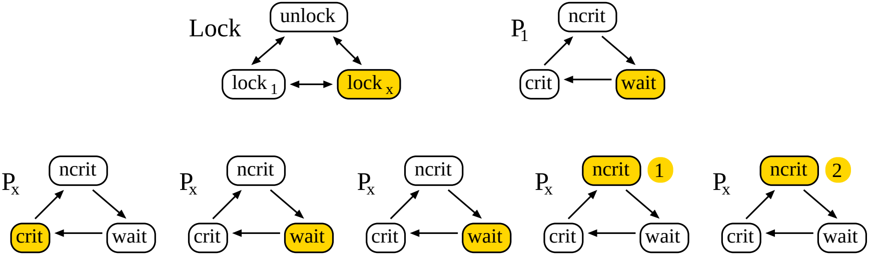


Improve Scalability by using Symmetry Reduction

Symmetry reduction of PRSIM cannot be used

- *n* processes are obviously symmetric
- but their index is used in the lock process
- need to adapt the model manually

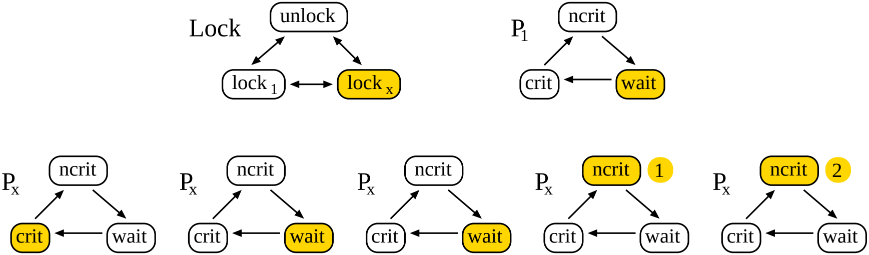
Using a generic representative



state counters:

crit : 1	ncrit 1 : 1
wait : 2	ncrit 2 : 1

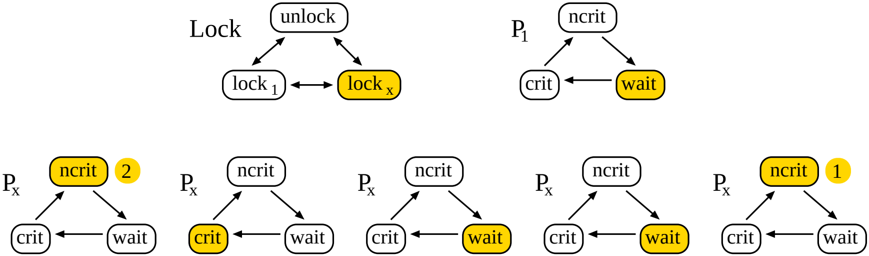
Using a generic representative



state counters:

crit : 1	ncrit 1 : 1
wait : 2	ncrit 2 : 1

Using a generic representative



state counters:

crit : 1	ncrit 1 : 1
wait : 2	ncrit 2 : 1

Exploiting symmetry reduction

Change model using a generic representative

- keep P_1 process unchanged
- use a counter for each state of $n-1$ processes P_x
- adapt lock process

MRMC + custom DTMC generation

- avoid model generation bottleneck in PRISM
- custom program builds DTMC of the model as sparse matrix
- MRMC operates directly on that matrix
- MRMC computes some of our properties (but not yet all)

Outline

Introduction

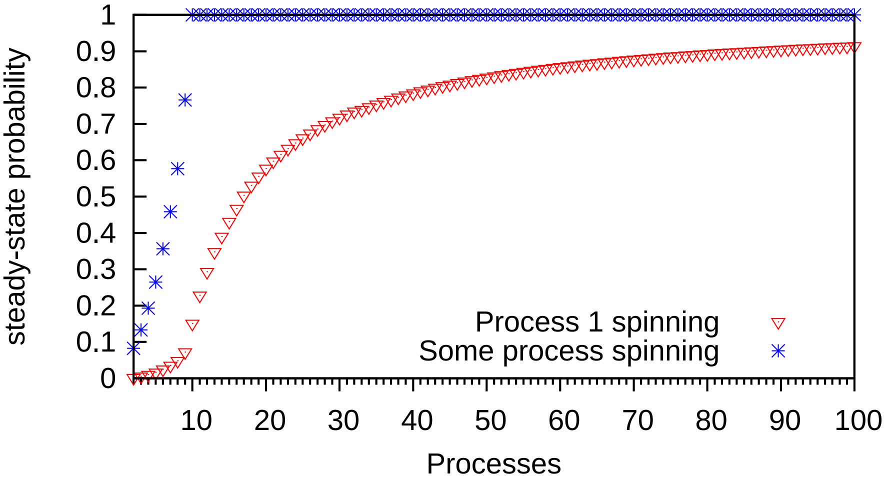
A test-and-test-and-set Lock and its DTMC model

Symmetry Reduction

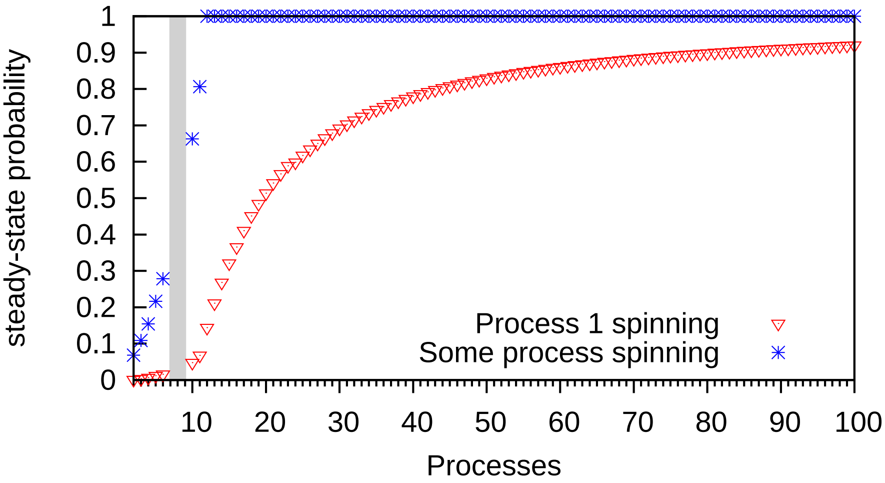
Results for the reduced model

Conclusion

Spinning probability for [40, 50]

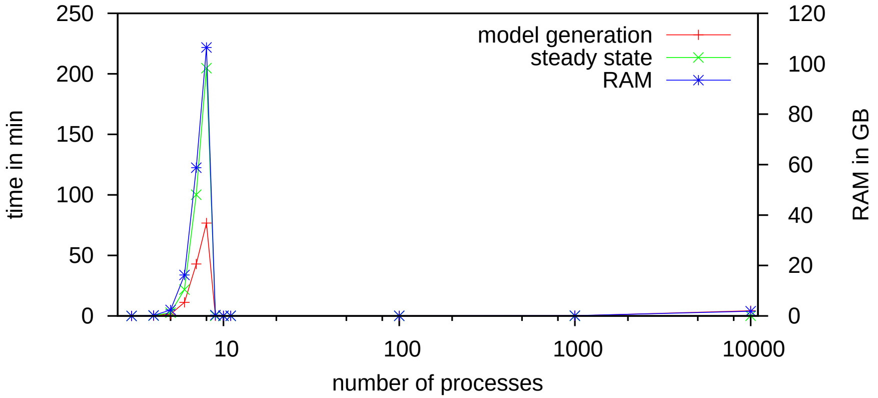


Spinning probability for [50, 60]



model generation fails for 7–9 processes because of a 190 GB RAM limit

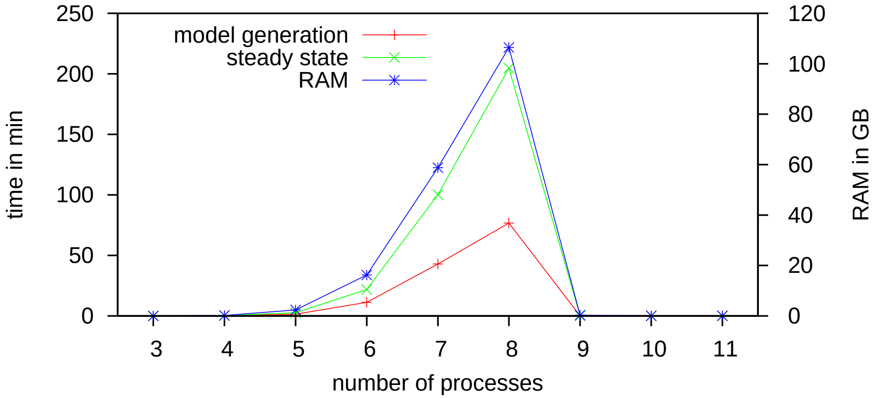
Scalability for MRMC, Distribution [40, 50]



state numbers:

8	387,320,107	100	205,637
9	1,211,760	1,000	334,337
10	189,311	10,000	1,621,337

Scalability for MRMC, Distribution [40, 50]



state numbers:	8	387,320,107	100	205,637
	9	1,211,760	1,000	334,337
	10	189,311	10,000	1,621,337

Why does it scale so extremely well?

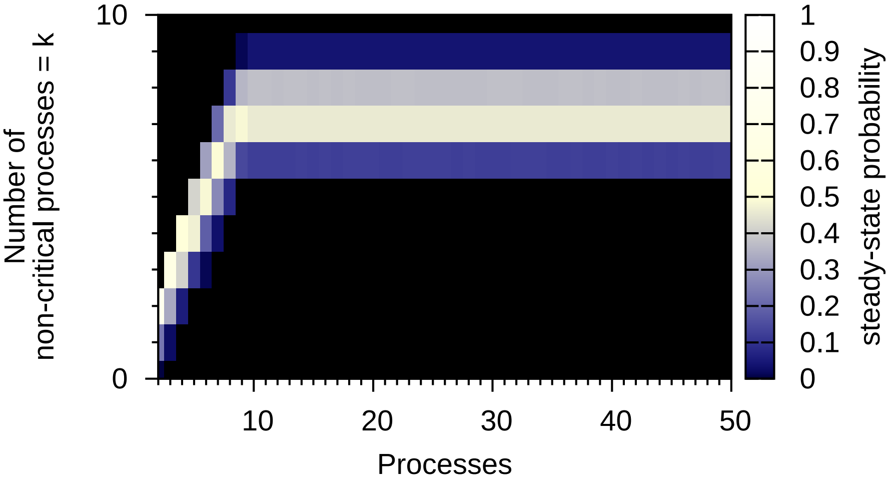
Lock is oversaturated

- 1 process is in the critical section
- 6–9 processes are in their non-critical section
- the remaining processes spin
- adding another process only increases the spinning-counter by 1

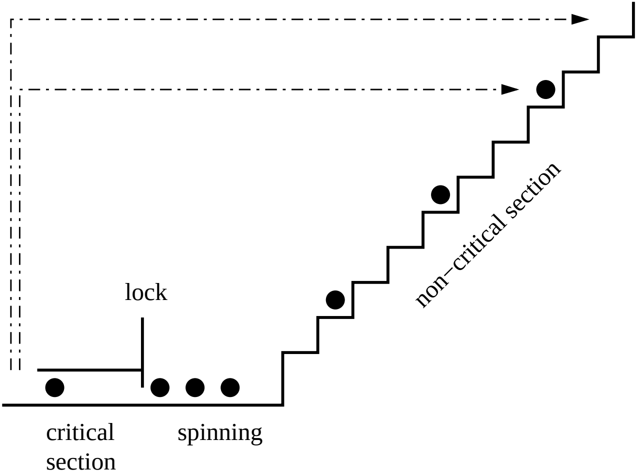
Processes in non-critical region show a regular pattern

- 1 process releases the lock (circa) every 6 time units
- chooses a non-critical waiting time of 40 or 50 time units
- distance of waiting time is regular

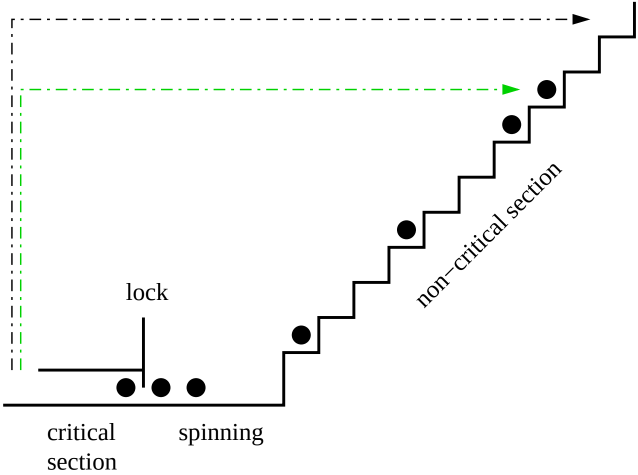
Processes in non-critical region



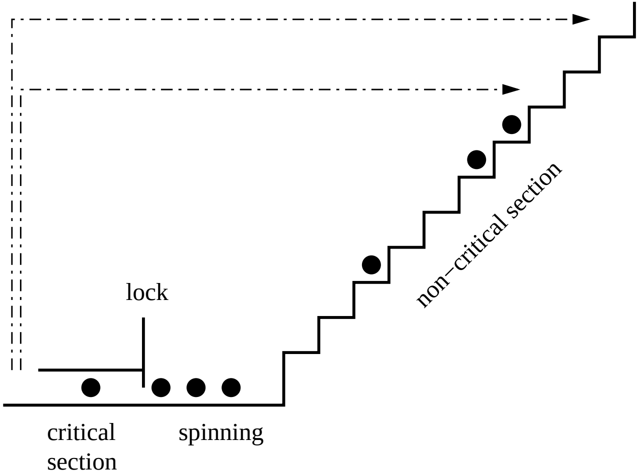
Ladder effect in non-critical region



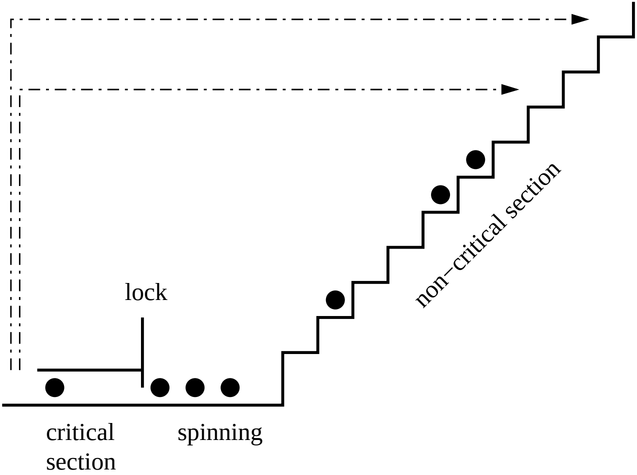
Ladder effect in non-critical region



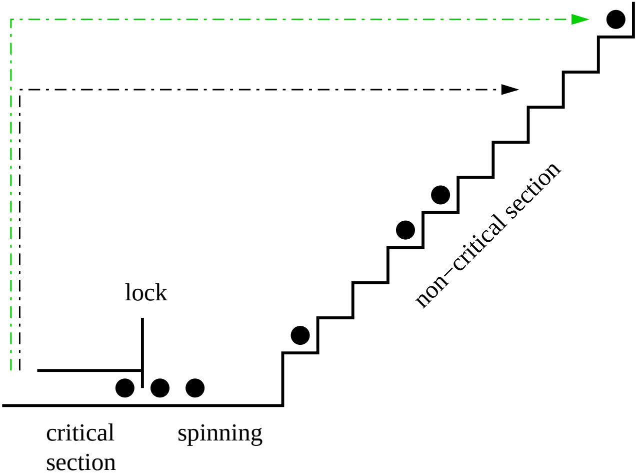
Ladder effect in non-critical region



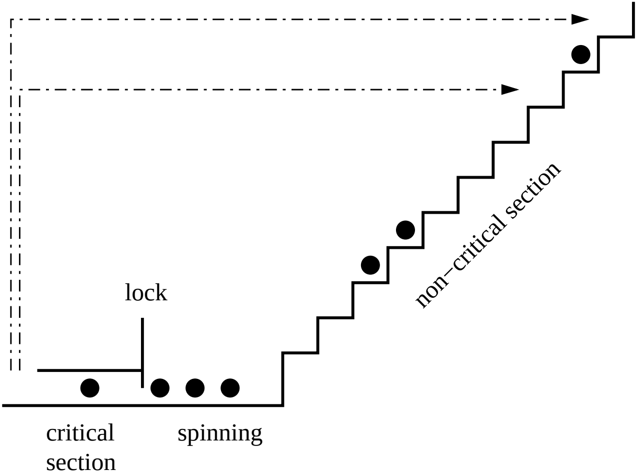
Ladder effect in non-critical region



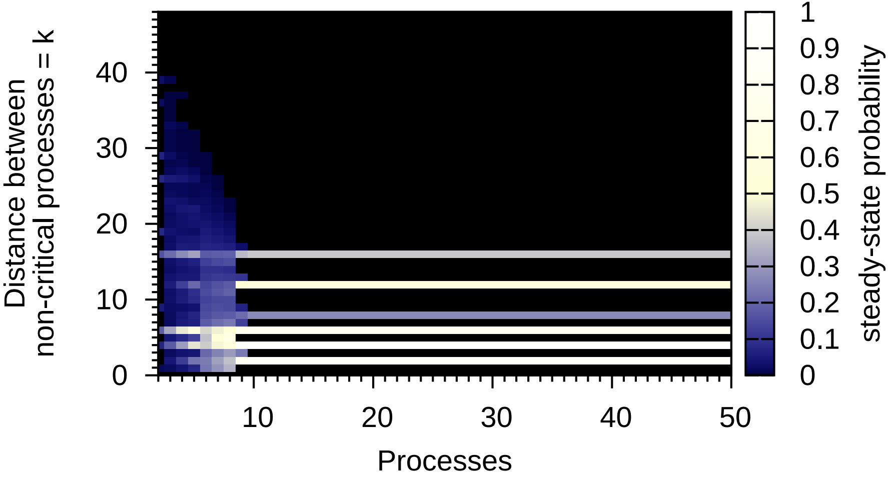
Ladder effect in non-critical region



Ladder effect in non-critical region



Ladder effect in non-critical region



Outline

Introduction

A test-and-test-and-set Lock and its DTMC model

Symmetry Reduction

Results for the reduced model

Conclusion

Conclusion

Improved scalability of Spin Lock model

- model specific symmetry reduction
- using MRMC (to avoid the model generation bottleneck in PRISM)
- scales up to $10,000$ processes
- scalability is linked to the over-saturation of the lock

A spin lock for 10,000 processes?

- certainly nonsense, but
- overbooked services exist
- symmetry reduction will yield similar improvements there