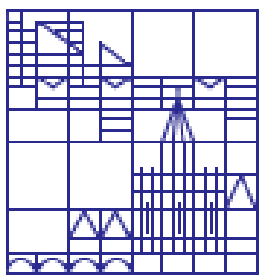
Directed Explicit-State Model Checking

Stefan Leue

University of Konstanz
Chair for Software Engineering

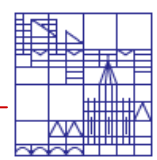
Stefan.Leue@uni-konstanz.de
http://www.inf.uni-konstanz.de/~soft

1 November 2004



University of Konstanz
Computer and Information Science

- **joint work with**
  - Stefan Edelkamp
    - Dortmund
  - Alberto Lluch-Lafuente
    - Pisa



Overview

1. Introduction

2. State Space Exploration Strategies

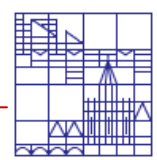
3. Directed Explicit-State Model Checking for Safety Properties

4. Directed Explicit-State Model Checking for Liveness Properties

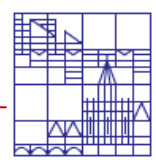
5. Directed Explicit-State Model Checking and Partial-Order Reduction

6. Current and Future Research

7. Conclusion



Overview



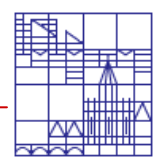
Motivation

- **Software Ubiquity**
  - software is a determining, if not the most determining, technology used in most technical systems used or developed

"*Our civilization runs on software*"

Bjarne Stroustrup

- **Software Complexity of Reactive, Embedded Systems**
  - ever growing complexity of services provided
    - bugs as ubiquitous as these systems are becoming

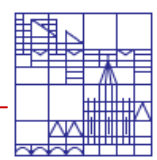


Motivation

- **Economical impact ...**
  - software failures cost huge amounts of money
    - according to NIST study (2002), software failures in the US cost about $59.5 billion annually (0.6% of US GDP)
    - esitmate that $22.2 billion could be saved by improved software quality assurance (including testing)

Quoted from *Software disasters are often people problems*, http://www.msnbc.msn.com/id/6174622/

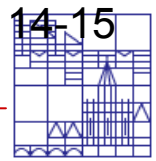
- **... but it's also an ehtics issue**
  - threat to human well-being and life
    - baby killed by front-seat airbag, even though airbag had been software disabled
    - re-use of a bit in the assembler code suspected as cause



Motivation

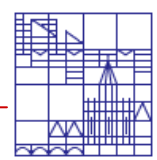
- **Faults are often due to unexpected interleavings**
  - AT&T Telephone Switch Failure, January 15, 1990
    - failure: new software release in 4ESS switches
      - a switch would no longer send messages to indicate it is no longer faulty, but other switches would notice this due to resumed activity
      - one switch signaled it went down, went through recovery cycle (reboot), and resumed sending traffic
      - second switch accepted down message and attempted to reset
      - due to software fault, when second switch received second message from first switch, it could not properly handle it and went itself through recovery / resume cycle
      - fault propagated through 114 different switches in AT&T network (all ran the same software!)
    - result: 9 hr. nationwide telephone traffic blockade

Source: Peter G. Neumann, *Computer Related Risks*, ACM Press, 1995, p. 14-15



Motivation

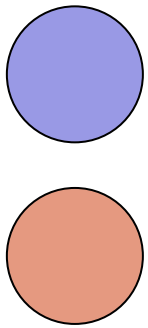
- **Model Checking**
  - automated, systematic state space exploration
  - provides error traces
  - a complementing technique
    - code reviews / inspections find most obvious faults
    - testing finds hidden, but unanticipated faults
    - model checking aims at unanticipated faults

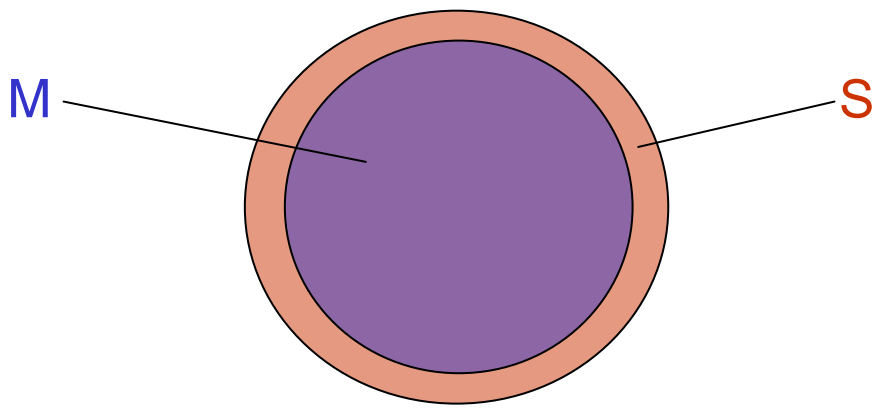


Model Checking

- **What is Model Checking?**
  - basic principle
    - given
      - (software-) model M
      - property specification S
    - does M satisfy S?

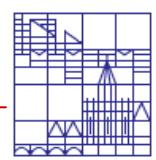
      M |= S
      - that is the case if every behaviour of M is also a behaviour of S
      - i.e., the model M does not reveal behaviour violating the specification S





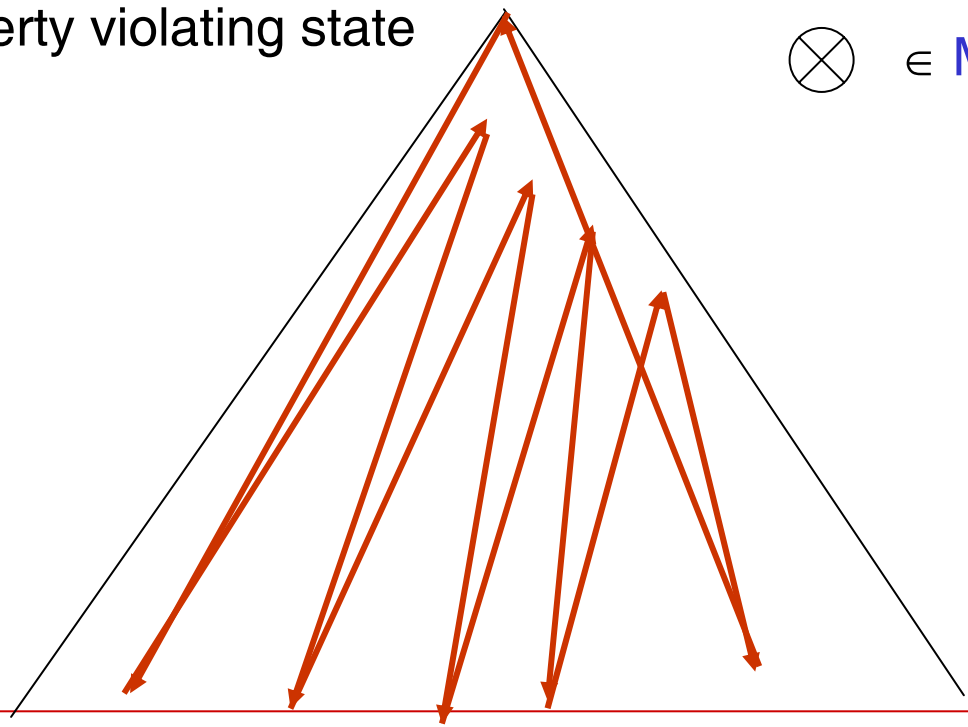
Model Checking

- **Explicit State Model Checking**
  - state space given by explicit state-transition function
  - successful in the verification of concurrent, asynchronous software systems
    - automatic, terminating procedure if M finite
    - error explanation by output of offending execution paths
      - "error trails"
      - "counterexamples"
    - partial order reduction, bit-state hashing to counter state space explosion
  - one prominent example
    - SPIN (G. Holzmann, formerly Bell Labs, now JPL)
      - ACM Software Systems Award



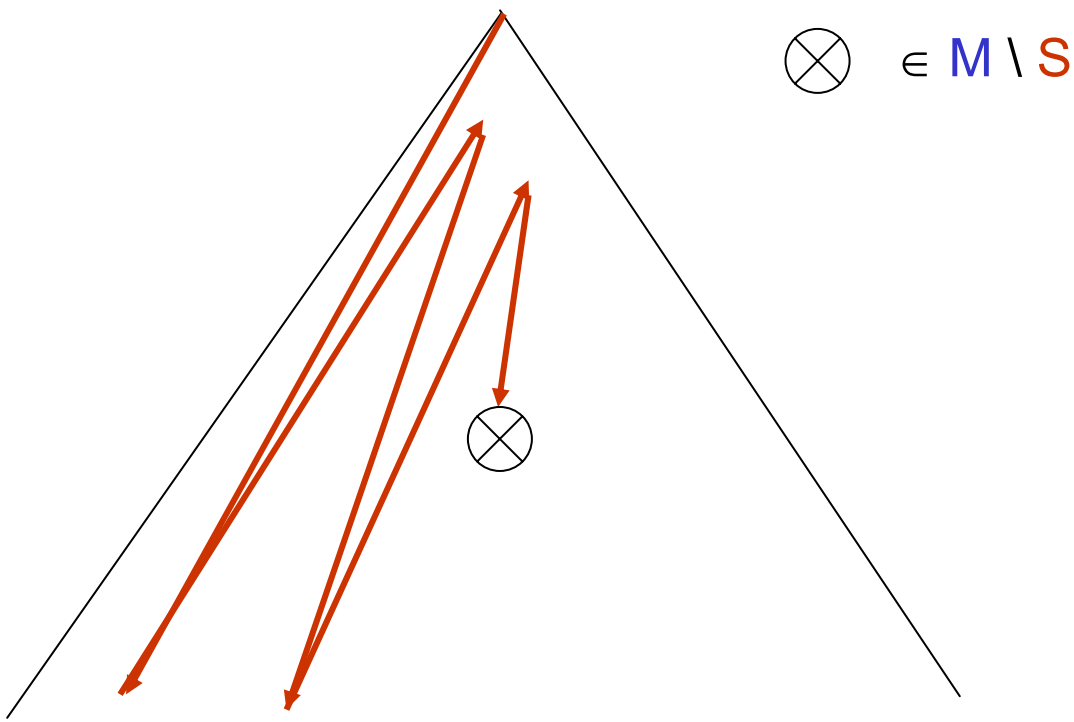
Model Checking of Safety Properties

- **Classical Use: Verification**
  - safety: invariants, absence of deadlock, reachability of states,...
  - DFS in the system's global state space
    - if no property violating state found
      - termination after complete state space exploration
    - if property violating state found
      - output counterexample
        - search stack contains path from initial system state into property violating state



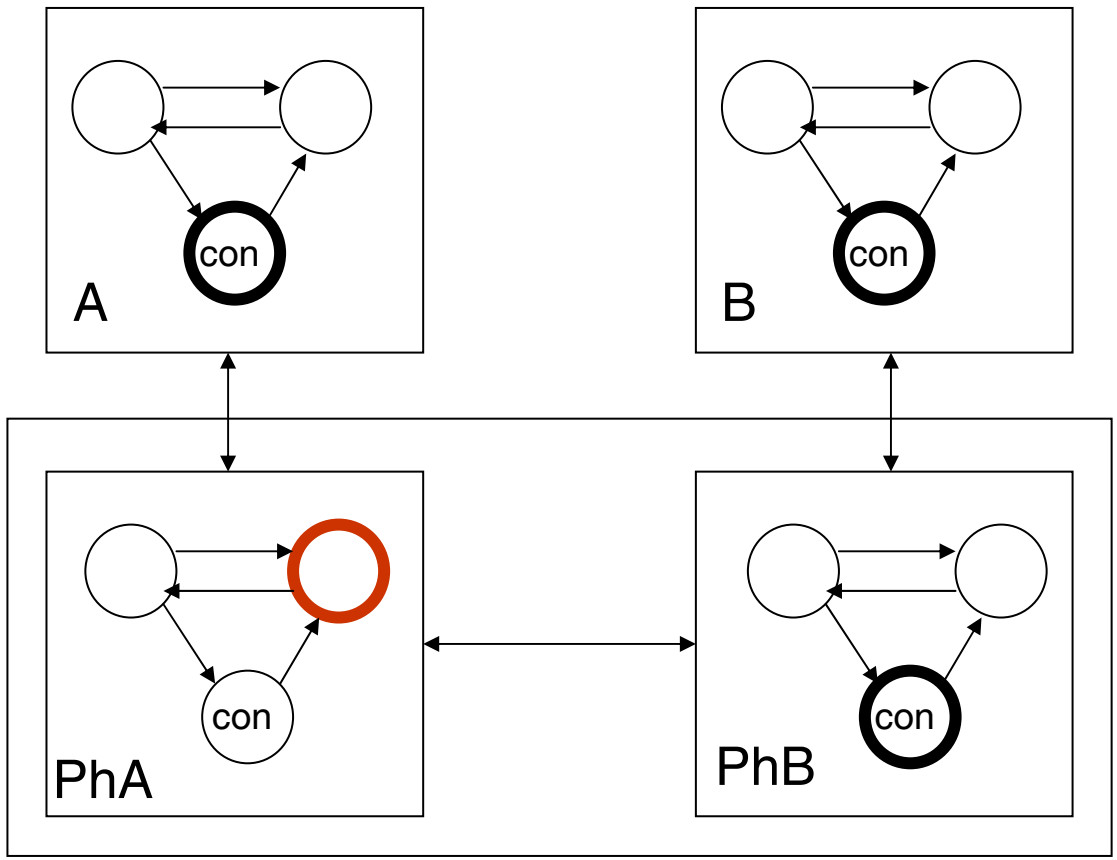
Model Checking of Safety Properties

- **More Recently: Use of Model Checker for Debugging Purposes**
  - let p an undesired state (safety) property
  - claim □¬p
    - model checker will try to disprove this claim and find a trail into a state satisfying the undesired property
  - error explanation



Model Checking of Safety Properties

- **Example: Debugging of POTS**

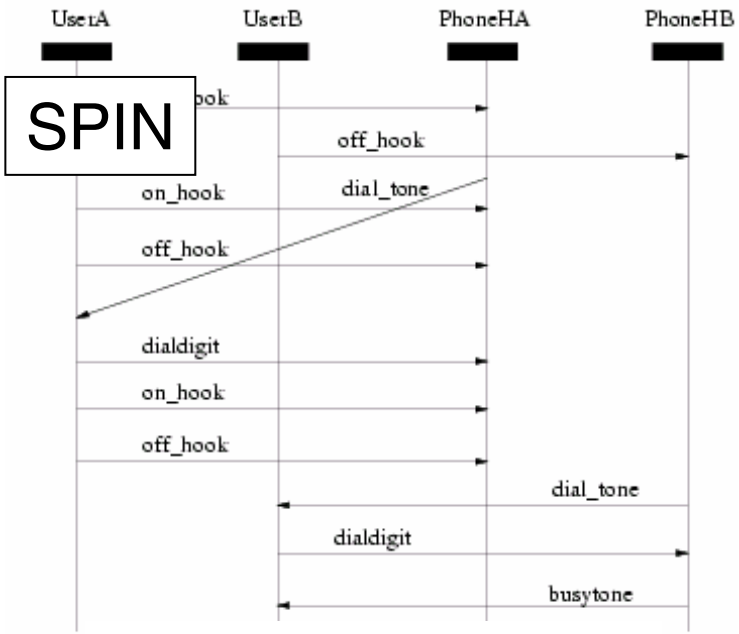


Invariant Property

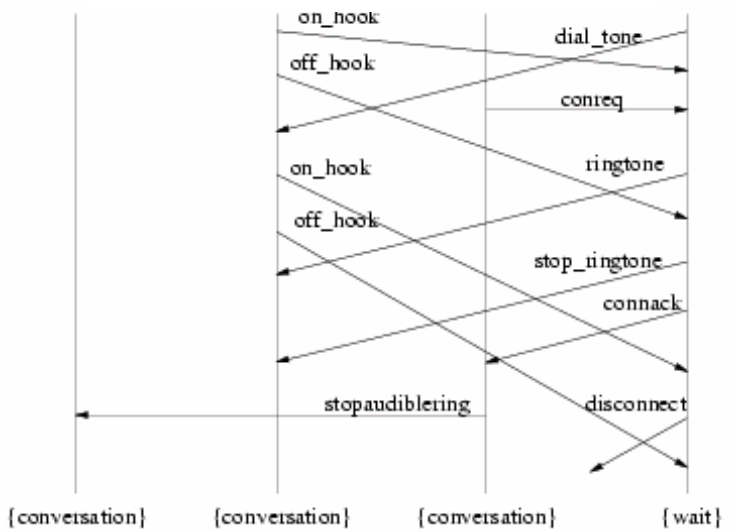
$$\neg(A@con \wedge \neg PhA@con \wedge B@con \wedge PhB@con)$$

Counterexamples

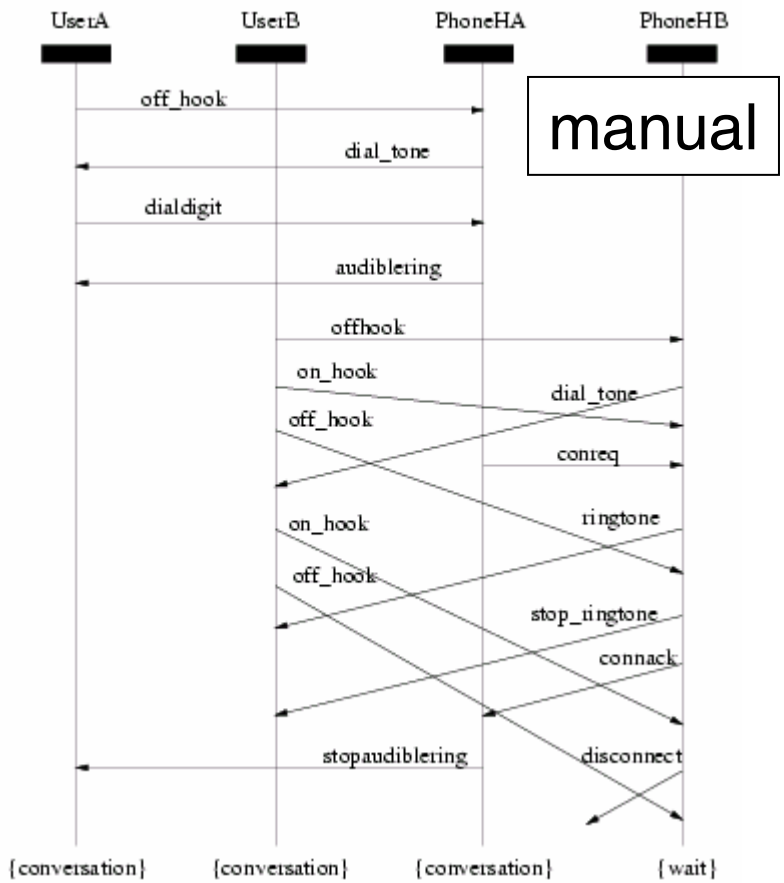
SPIN



… 440 messages …

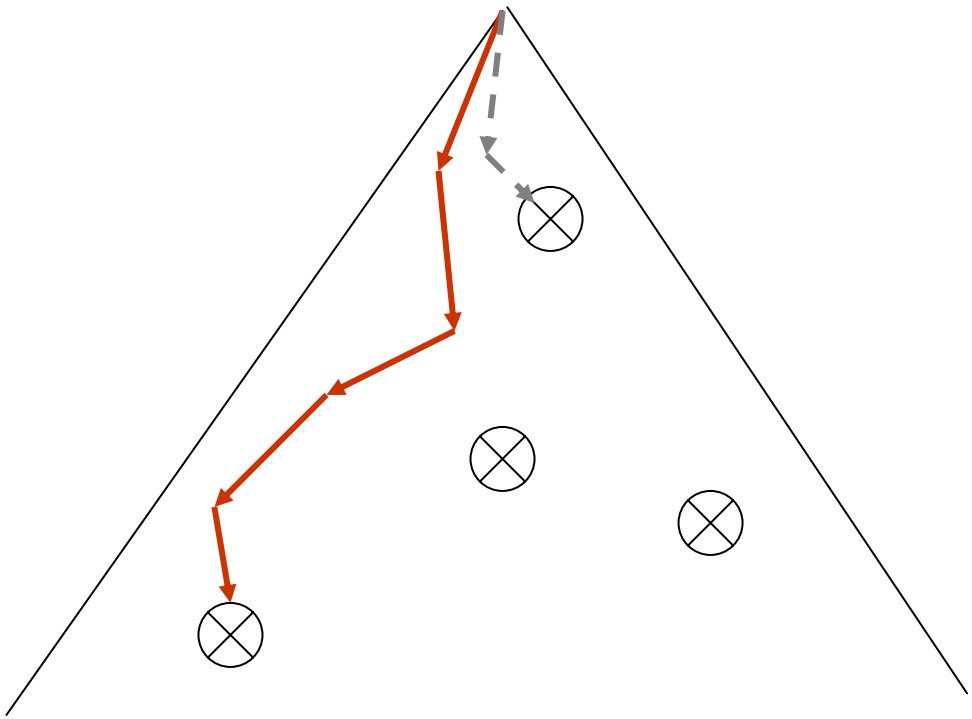


manual



total of 16 messages

Counterexample with Depth-First Search (DFS)



- **Shorter Counterexamples**
  - shorter path to **the same** property-violating state
  - shorter path to **some** property-violating state
- **State Space Exploration Strategies?**

Overview

1. Introduction

2. State Space Exploration Strategies

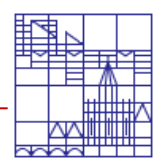
3. Directed Explicit-State Model Checking for Safety Properties

4. Directed Explicit-State Model Checking for Liveness Properties

5. Directed Explicit-State Model Checking and Partial-Order Reduction

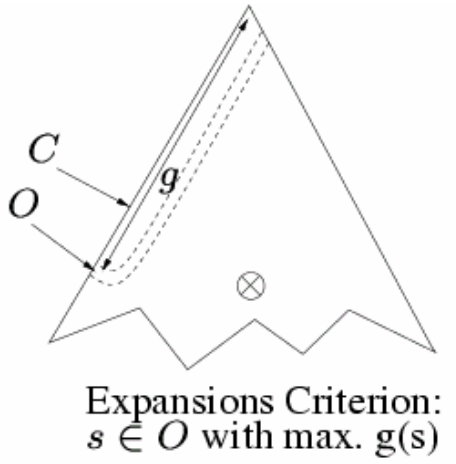
6. Current and Future Research

7. Conclusion



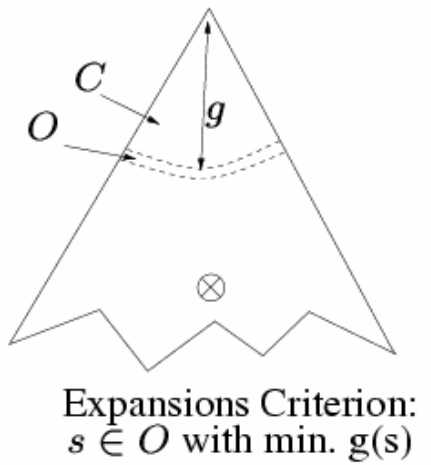
Uninformed Search

Depth-First Search (DFS)



- memory efficient
- potentially long error trails
- stack-based

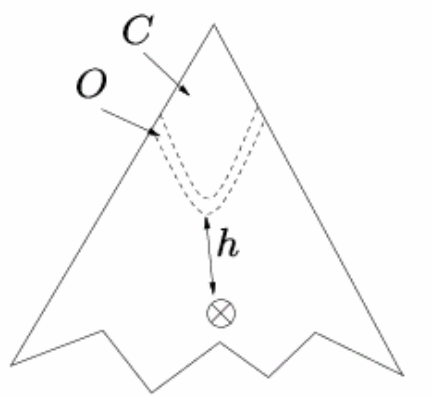
Breadth-First Search (BFS)



- memory inefficient
- shortest error trails
- no stack

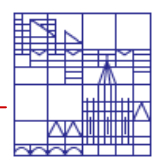
Informed Search

Best-First Search (BF)



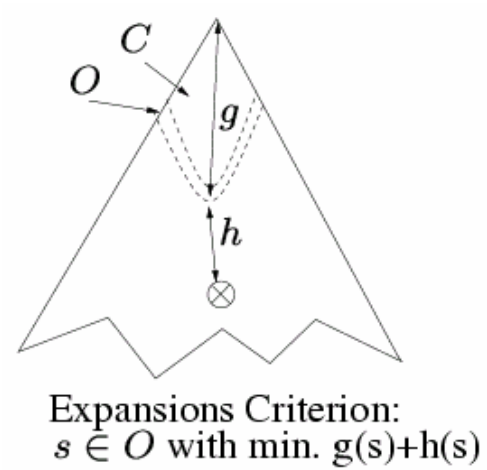
- informed search
  - heuristic estimate h(S): estimate of length of path to goal state S
- no guarantee for optimally short trails (sub-optimal solution)
- complete algorithm
  - goal state, if present, will be found
- often improved error trails

Graphiken von A. Lluch-Lafuente



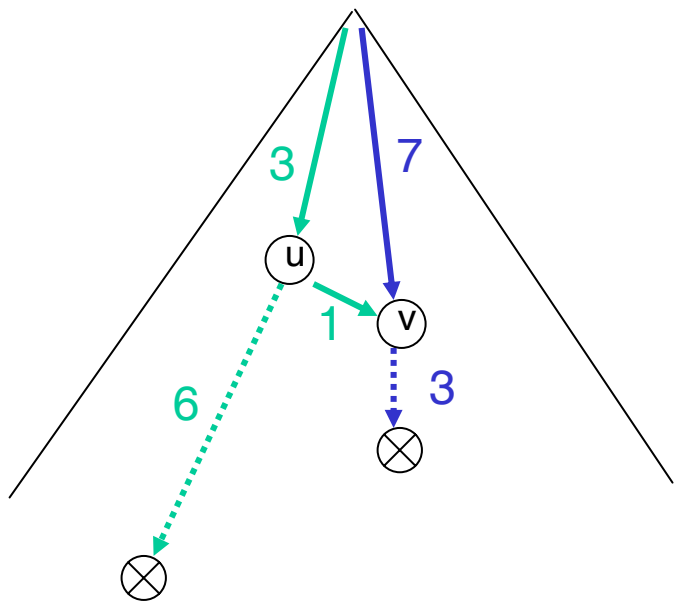
Informed Search

A*



- informed search
- optimally short error trails if h is a lower bound (admissible heuristics)
- complete
- no stack

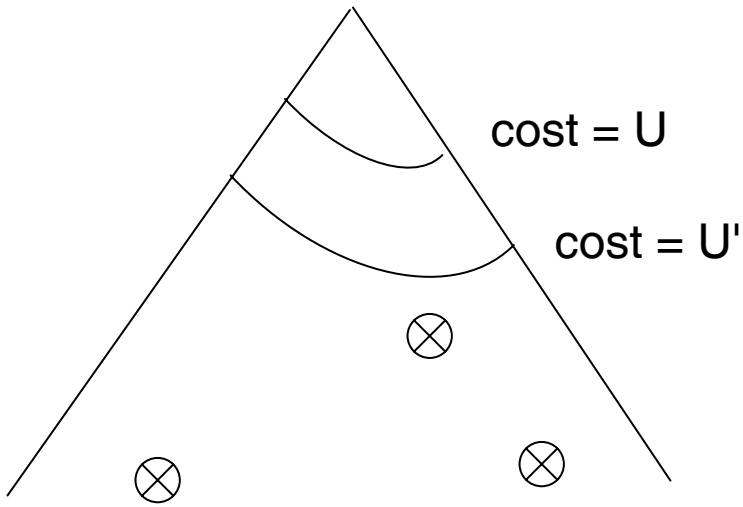
- re-opening of nodes
  - move states from *Closed* to *Open* if they can be reached on a shorter path
    - re-open v



Graphiken von A. Lluch-Lafuente

Informed Search

Iterative Deepening A* (IDA*)



- informed search
- iterate search depth according to increasing total cost bounds
- complete and optimal (simulates A*)
- stack-based DF search
  - bitstate hashing can be applied
    - avoid re-opening since generating path length information may be false due to duplicates
    - no problem for monotone heuristic estimates

Overview

1. Introduction

2. State Space Exploration Strategies

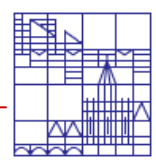
3. Directed Explicit-State Model Checking for Safety Properties

4. Directed Explicit-State Model Checking for Liveness Properties

5. Directed Explicit-State Model Checking and Partial-Order Reduction

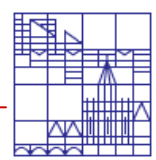
6. Current and Future Research

7. Conclusion



Directed Explicit-State Model Checking

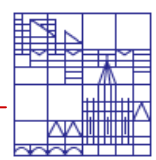
- **Statespace Exploration**
  - goal: locating errors
    - short error trails for easy error interpretation
  - exploration / search algorithms
    - BF
    - A*
      - A* does not maintain search stack
        - book-keeping to preserve predecessor information
    - IDA*
- **Tool**
  - implemented in HSF-SPIN (Lluch-Lafuente / Edelkamp, Freiburg)
    - based on SPIN version 3.4
    - handles a large fragment of Promela
    - http://www.informatik.uni-freiburg.de/~lafuente/hsf-spin/



Heuristic Estimator Functions

- **Formula based Heuristic H_f**
  - let f a state formula, S a global system state
    - H_f is a formula interpreted over S yielding a (lower bound) estimate for number of steps necessary to reach state satisfying f from S
  - for queue expressions and relational operators:

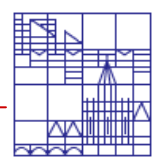
f	$H_f(S)$	$\overline{H}_f(S)$
$full(q)$	$capacity(q) - length(q)$	if $full(q)$ then 1, else 0
$empty(q)$	$length(q)$	if $empty(q)$ then 1, else 0
$q?[t]$	length of minimal prefix of q without t (+1 if q lacks message tagged with t)	if $head(q) \neq t$ then 0, else maximal prefix of t's
$a \otimes b$	if $a \otimes b$ then 0, else 1	if $a \otimes b$ then 1, else 0



Heuristic Estimator Functions

- **Formula based Heuristic H_f**
  - for predicates over control state locations
    - example for f: PhA@con
      - process i in control state s
    - lower bound for number of global state transitions is number of local state transitions
    - compute all-pairs-shortest path for local state transition matrix D_i

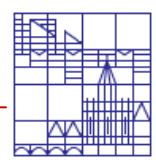
f	$H_f(S)$	$\overline{H}_f(S)$
$i@s$	$D_i(pc_i, s)$	if $pc_i = s$ 1, else 0



Heuristic Estimator Functions

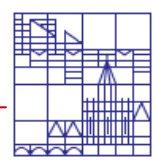
- **Formula based Heuristic H_f**
  - for boolean expressions

f	$H_f(S)$	$\overline{H}_f(S)$
$true$	0	∞
$false$	∞	0
a	**if** a **then** 0 **else** 1	**if** a **then** 1 **else** 0
$\neg g$	$\overline{H}_g(S)$	$H_g(S)$
$g \vee h$	$\min\{H_g(S), H_h(S)\}$	$\overline{H}_f(S) + \overline{H}_g(S)$
$g \wedge h$	$H_g(S) + H_h(S)$	$\min\{\overline{H}_g(S), \overline{H}_h(S)\}$



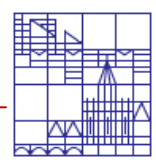
Heuristic Estimator Functions

- **Absence of Deadlock**
  - active processes heuristic: $H_{ap}(S)$
    - computes the number of currently active (=non-blocked) processes
    - relatively small range of heuristic function
  - H_U
    - based on *user* characterization of control states as "dangerous"
    - can be highly informative



Heuristic Estimator Functions

- **Assertions**
  - transition `(u, assert(a), v)` in process i
    - $f = i@u \wedge \neg a$
    - apply H_f
- **Partially or Completely Known State Vectors**
  - trail-guided model checking
    - characterize target state by given error trail
      - is there a shorter trail to same (or equivalent) state
    - directed search based on Hamming-distance heuristics
      - not always admissible but yields good results
        - admissible if each transition changes at most one bit in the state vector



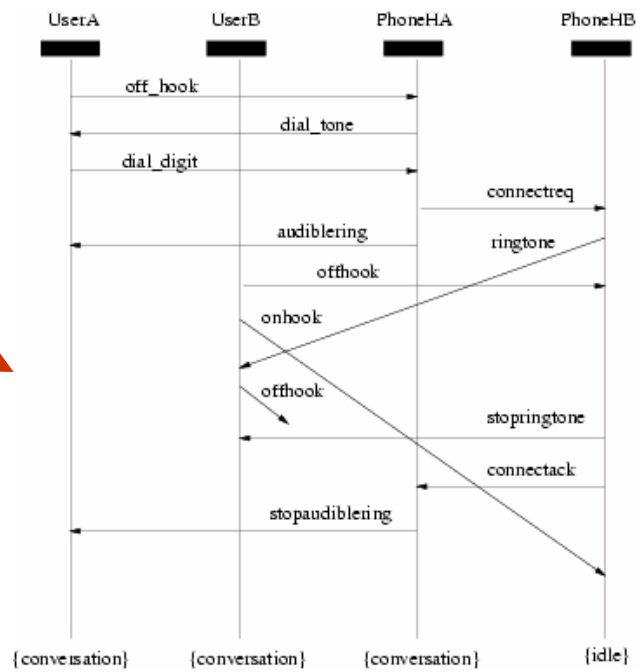
HSF-SPIN Experiments

- **Invariant Violation (H_f)**

Elevator	BFS	DFS	A*	BF	SPIN
s	38,662	279	38,598	2,753	292
e	38,564	279	38,506	2,297	292
t	160,364	356	160,208	5,960	348
l	203	510	203	421	510

states
exp.states
transitions
err. **l**ength

POTS	BFS	DFS	A*	BF	SPIN
s	24,546	-	6,654	781	148,049
e	17,632	-	3,657	209	148,049
t	99,125	-	18,742	1,067	425,597
l	81	-	81	83	2,765



12 messages

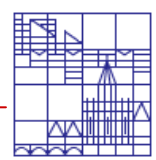
HSF-SPIN Experiments

- **Deadlock Detection with H_{ap}**

states
exp.states
transitions
err. **l**enght

GIOP	BFS	DFS	A*	BF	SPIN
s	40,847	218	31,066	117	326
e	37,266	218	27,061	65	326
t	151,671	327	108,971	126	364
l	58	134	58	65	134

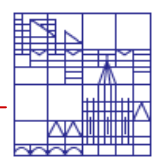
Philosophers	BFS	DFS	A*	BF	SPIN
s	3,678	1,341	67	493	1,341
e	2,875	1,341	17	225	1,341
t	15,775	1,772	73	622	1,772
l	34	1,362	34	66	1,362



HSF-SPIN Experiments

- **Dining Philosophers**
  - finding of errors where DFS fails
    - BFS up to p = 8,
    - DFS (SPIN) up to p = 15
    - for A*: p ≥ 255 (linear behavior dependent on p)
  - SPIN: DFS following lexical ordering of processes
  - A*: directed search avoids pitfalls of lexically ordered exploration

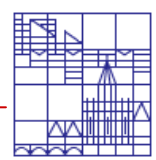
p		BFS	DFS	A*	BF	SPIN
2	s	9	6	6	6	6
	e	7	6	6	4	6
	t	10	7	4	6	7
	l	10	10	10	10	10
3	s	19	19	12	26	19
	e	14	10	7	23	10
	t	29	12	13	43	12
	l	14	18	14	14	18
4	s	56	45	19	70	45
	e	42	45	9	57	45
	t	116	62	21	142	116
	l	18	54	18	26	54
8	s	3,768	1,341	67	493	1,341
	e	2,875	1,341	17	225	1,341
	t	15,775	1,772	73	622	1,772
	l	34	1,362	34	66	1,362
14	s	-	-	199	1,660	2,164,280
	e	-	-	29	1,963	2,164,280
	t	-	-	211	684	27,050,400
	l	-	-	58	114	9,998
16	s	-	-	259	2,201	-
	e	-	-	33	893	-
	t	-	-	273	2,578	-
	l	-	-	66	130	-



HSF-SPIN Experiments

- **IDA* and Double-Bitstate Hashing**
  - deadlock detection in GIOP
  - memory bound: 256 MB, time limit 120 mins
  - IDA* and double bit-state hashing
    - number of expansions

Depth	A*	IDA*	IDA*+bitstate
58	150,344	146,625	146,625
59	168,191	164,982	164,982
60	184,872	184,383	184,383
61	-	206,145	206,145
62	exceeds memory limit	exceeds time limit	229,626
63	-	-	255,411
64	-	-	282,444
65	-	-	311,340
66	-	-	341,562
67	-	-	373,422
68	-	-	407,310
69	-	-	442,941
70	-	-	goal found



HSF-SPIN Experiments

- **Trail Improvement with Hamming (H^e_d) and FSM (H^e_m) Distance Metrics**

`marriers(4)`

	DFS	A*			
		f,H^e_d	f,H^e_m	f_e,H^e_d	f_e,H^e_m
s	407,009	26,545	225,404	126,479	1,754,408
l	121	99	66	121	121
m	58 MB	8 MB	26 MB	22 MB	248 MB
r	367 s	6 s	126 s	62 s	1,150 s

`pots`

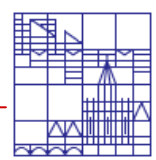
	DFS	A*			
		f,H^e_d	f,H^e_m	f_e,H^e_d	f_e,H^e_m
s	118,099	988	13,865	4,432	14,714
l	987	89	81	89	88
m	58 MB	7 MB	11 MB	7 MB	12 MB
r	81 s	5 s	5 s	5 s	5 s

`leader(8)`

	DFS	A*			
		f,H^e_d	f,H^e_m	f_e,H^e_d	f_e,H^e_m
s	36	10,733	3,161	10,773	3,173
l	71	71	69	71	71
m	3 MB	10 MB	6 MB	10 MB	6 MB
r	1 s	6 s	1 s	6 s	1 s

`giop(2,1)`

	DFS	A*			
		f,H^e_d	f,H^e_m	f_e,H^e_d	f_e,H^e_m
s	218	988	30,629	23,518	446,689
l	134	67	65	134	134
m	3 MB	5 MB	22 MB	18 MB	266 MB
r	1 s	1 s	8 s	21 s	128 s



Overview

1. Introduction

2. State Space Exploration Strategies

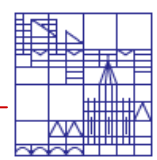
3. Directed Explicit-State Model Checking for Safety Properties

4. Directed Explicit-State Model Checking for Liveness Properties

5. Directed Explicit-State Model Checking and Partial-Order Reduction

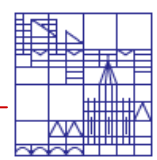
6. Current and Future Research

7. Conclusion



Directed Model Checking - Liveness

- **Liveness Properties**
  - satisfied along a path of infinite length
  - ”every sending is followed by a reception“ (response)
    - □(send ⇒ ◊receive)
      - contains liveness component
  - amounts to recognizing ω-regular expressions of the form uv^{ω}
    - => cycle detection

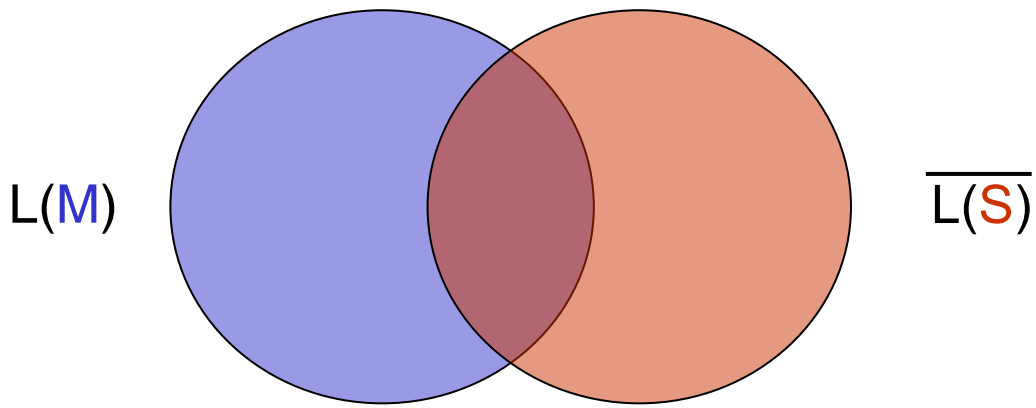


Directed Model Checking - Liveness

- **Automata-based Model Checking for Liveness Properties**
  - does model M satisfy the specification S

    $M \models S$
  - M satisfies S iff

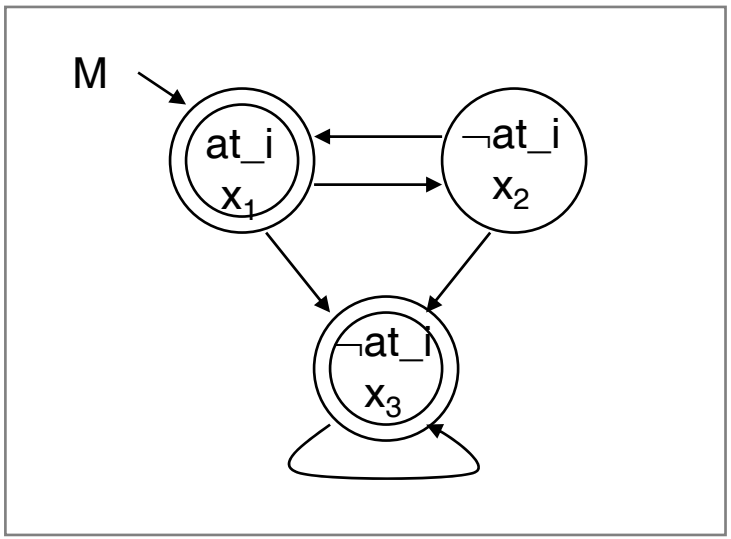
    $L(M) \subseteq L(S) \Leftrightarrow L(M) \cap (L(S) = \varnothing$

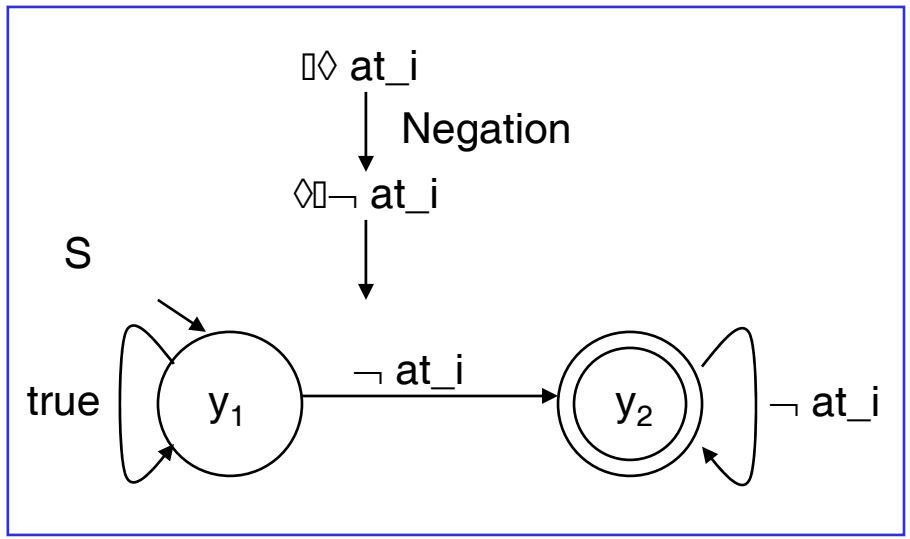


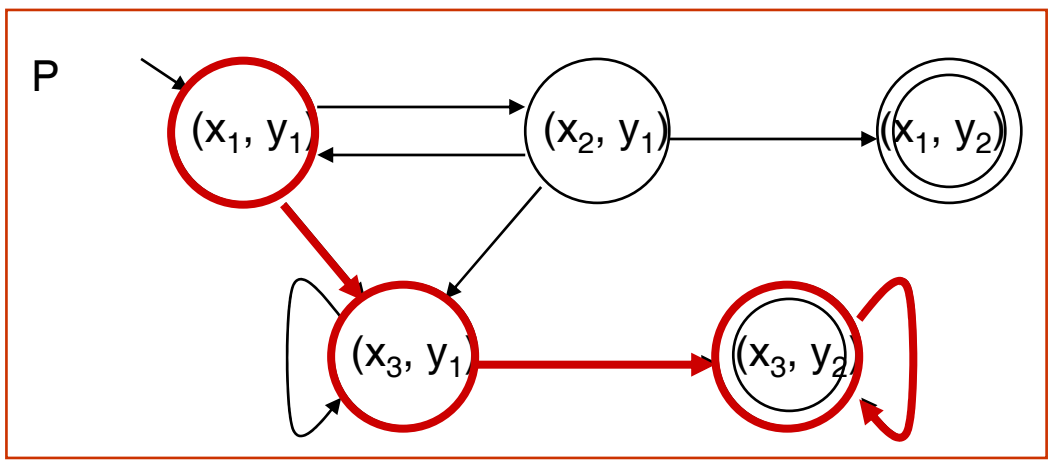
  - automata based emptyness test for $L(M) \cap \overline{L(S)}$ after Vardi and Wolper
    - detecting cycles in the synchronous product of M und $\overline{S}$

Directed Model Checking - Liveness

- **Automata-based Model Checking (after Vardi/Wolper)**
  - synchronous product construction





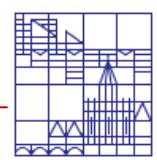


Directed Model Checking - Liveness

- **Exploration Algorithm**
  - nested DFS (NDFS)

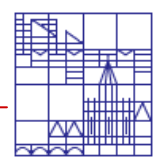
```
Nested-DFS(s)
    hash(s)
    for all successors s' of s do
      if s' not in the hash table then Nested-DFS(s')
    if accept(s) then Detect-Cycle(s)

Detect-Cycle(s)
    flag(s)
    for all successors s' of s do
      if s' on Nested-DFS-Stack then exit LTL-Property violated
      else if s' not flagged then Detect-Cycle(s')
```

  - note: second traversal in post-order
    - memory efficiency



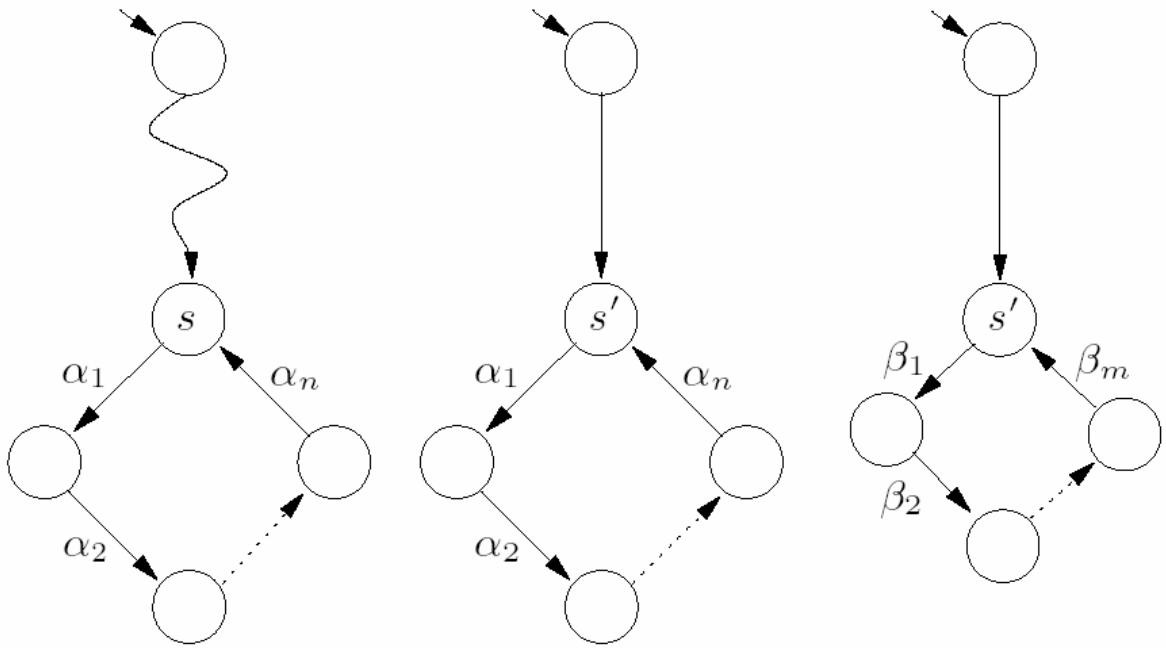
Directed Model Checking for Liveness

- **Potential of Improving Nested DFS**
  - improvement of second search (cycle finding) by directed search
    - not promising, since often only one cycle closing state on stack
  - alternative: pre-order traversal
    - quadratic time and linear memory overhead
  - improved nested DFS (I-NDFS)
    - results not as promising as for safety properties



Trail Improvement for Liveness

- **Trail Improvement**
  - given error trail
    - determine some state s starting a cycle
    - determine cycle of transitions α_1 .. α_n leading from s to s
  - improvement
    - path from root to s (or an equivalent s')
    - shorter cycle β_1 .. β_m equivalent to α_1 .. α_n



Trail Improvement für Liveness

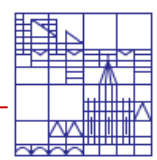
- **Trail Improvement**
  - m: Hamming distance
  - d: FSM distance
  - [s]: states equivalent to s
- **philo**
  - the existing cycle can be improved
- **GIOP**
  - search for equivalent state s' pays off

philo(8)

	NDFS	A*			
		s, H_d^s	s, H_m^s	$[s], H_d^{[s]}$	$[s], H_m^{[s]}$
s	57	18	11	18	11
l	71+68	31+4	31+4	31+4	31+4
m	3 MB	4 MB	4 MB	4 MB	4 MB
r	1 s	1 s	1 s	1 s	1 s

giop(2,1)

	NDFS	A*			
		s, H_d^s	s, H_m^s	$[s], H_d^{[s]}$	$[s], H_m^{[s]}$
s	7,331	355,249	931,901	1,730	1,929
l	288+2	288+2	288+2	52+2	52+2
m	8 MB	235 MB	609 MB	6 MB	6 MB
r	3 s	549 s	430 s	1 s	1 s



Overview

1. Introduction

2. State Space Exploration Strategies

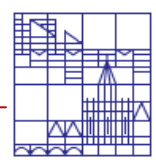
3. Directed Explicit-State Model Checking for Safety Properties

4. Directed Explicit-State Model Checking for Liveness Properties

5. Directed Explicit-State Model Checking and Partial-Order Reduction

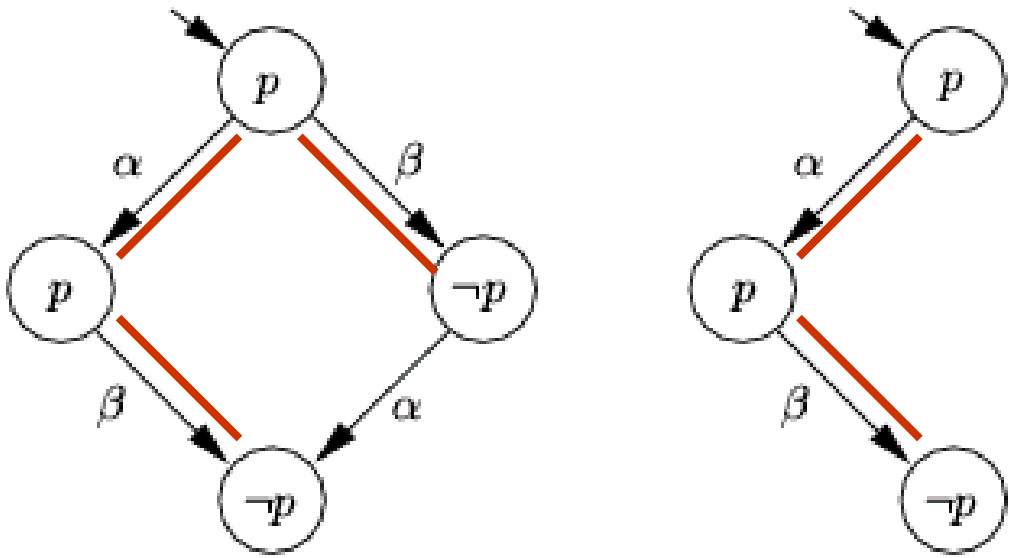
6. Current and Future Research

7. Conclusion



Partial Order Reduction

- **Partial Order Reduction**
  - exploit commutativity of concurrent transitions
    - exploration of only one representative of a class of equivalent paths in the state space
  - pre-processing: computing the "ample set" for a state
    - subset of the set of enabled transitions in a state that must be fully expanded
  - loss of solution length optimality



DMC and Partial Order Reduction

- **Conditions C0-C3 for the Construction of an Ample Set for Safety Protperties**

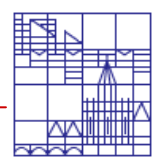
Condition C0: The ample set of s is void exactly when the enabled set of s is void.

Condition C1: Along every path in the full state space that starts at s, a transition that is dependent on a transition in $ample(s)$ cannot be executed without a transition in $ample(s)$ occurring first.

Condition C2: If a state s is not fully expanded, then each transition α in the ample set must be invisible with regard to P.

Condition C3: If for each state of a cycle, a transition α is always enabled, then α must be selected in the ample set of some of the states of the cycle.

  - conditions C0-C2 are independent of the used search algorithm



DMC and Partial Order Reduction

- **Conditions C0-C3 for the Construction of an Ample Set for Safety Protperties**

> **Condition C3**: If for each state of a cycle, a transition α is always enabled, then α must be selected in the ample set of some of the states of the cycle.

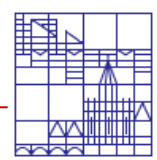
  - C3 can be over-approximated as $C3_{cycle}$:

    Every cycle in the reduced state space contains at least one state that is fully expanded.

    - expensive, since it requires cycle detection
  - C3 further over-approximated for safety properties as $C3_{stack}$ (for instance in SPIN):

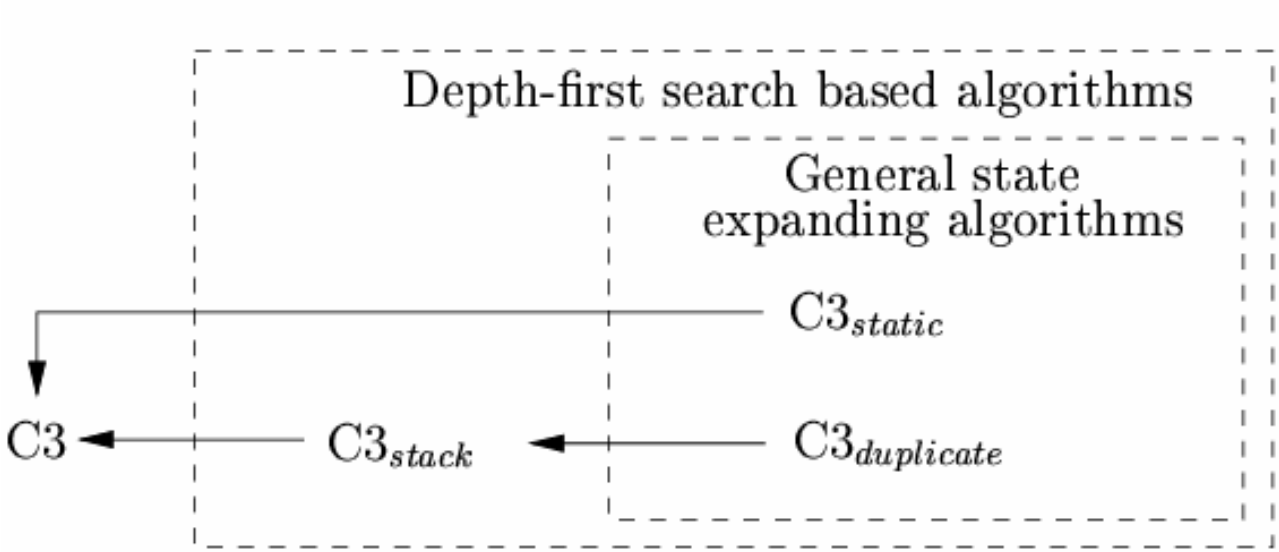
    If a state s is not fully expanded, then at least one transition in ample(s) does not lead to a state that is on the search stack

    - problem:
      - only applicable to stack-based DFS search
      - A* does not use a search stack



DMC and Partial Order Reduction

- **Overapproximations of C3 for non-DFS Algorithms**
  - $C3_{duplicate}$
    - *if a state is not entirely expanded then at least one transition in ample(s) does not lead into a **previously visited** state*
      - will not necessarily close a cycle
  - $C3_{static}$
    - *if a state is not fully expanded, then there is no transition in ample(s) which closes a cycle in the **control flow of a local process***

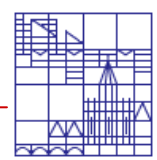


Experiments with HSF-SPIN

- **Safety Property Violation**
  - with po-reduction, with heuristic A* search

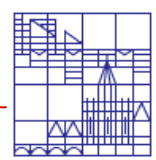
Model	Reduction	States	Transitions	Length
marriers	no	5,077	12,455	50
	$\mathbf{C3}_{duplicate}$	2,988	4,277	50
	$\mathbf{C3}_{static}$	1,604	1,860	50
pots	no	2,668	6,519	67
	$\mathbf{C3}_{duplicate}$	1,662	3,451	67
	$\mathbf{C3}_{static}$	1,662	3,451	67
leader	no	7,172	22,876	58
	$\mathbf{C3}_{duplicate}$	65	3,190	77
	$\mathbf{C3}_{static}$	399	3,593	66
giop	no	31,066	108,971	58
	$\mathbf{C3}_{duplicate}$	21,111	48,870	58
	$\mathbf{C3}_{static}$	12,361	24,493	58

  - further experiments with DMC + PO
    - no negative mutual influence
    - occasional synergetic effects were observed



Trail Reordering using PO Techniques

- **Trail Reordering**
  - loss of solution quality when process can immediately perform transition into error state but search algorithm postpones taking that transition
  - post-processing of error trail
    - ignore those transitions in the error trail that are *independent* from transition leading directly into error state
      - do not disable each other, are commutative
    - conditions to ignore *irrelevant* transition α
      - α is *invisible* wrt the property to be analyzed
      - α is *independent* from any later occurring transition
      - α *cannot enable* any later occurring transition
    - eliminate irrelevant transitions successively starting at the end of the trail
      - previously relevant transitions may become irrelevant, for instance, if they were only dependent on an eliminated transition



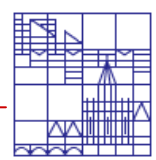
Trail Reordering using PO Techniques

- **Experimental Results**

leader(n)

n	$C3^{-}_{duplicate}$	reord.	$C3_{static}$	reord.	opt.
3	49	46	49	46	46
4	63	53	56	53	52
5	77	59	66	59	56
6	91	63	76	63	60
7	105	67	86	67	64
8	119	71	96	71	68

  - error states after reordering are very similar, differ only in transitions relevant to other processes than those which define the property (two processes assuming to be the leader)



Overview

1. Introduction

2. State Space Exploration Strategies

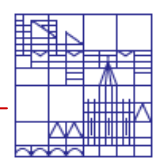
3. Directed Explicit-State Model Checking for Safety Properties

4. Directed Explicit-State Model Checking for Liveness Properties

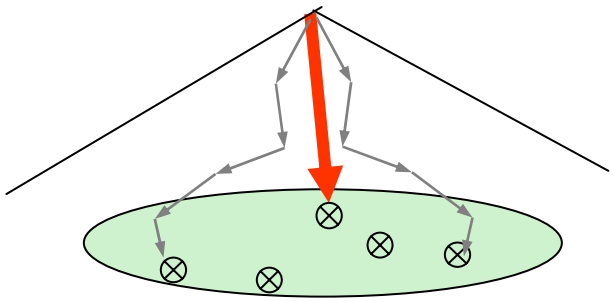
5. Directed Explicit-State Model Checking and Partial-Order Reduction

6. Current and Future Research

7. Conclusion



DMC and Probabilistic Model Checking



Set of "Goal" States

• satisfying reachability condition

• jointly contributing to required probability measure

CSL Model Checking

• establishes reachability and satisfaction of probability measure

• no counterexample

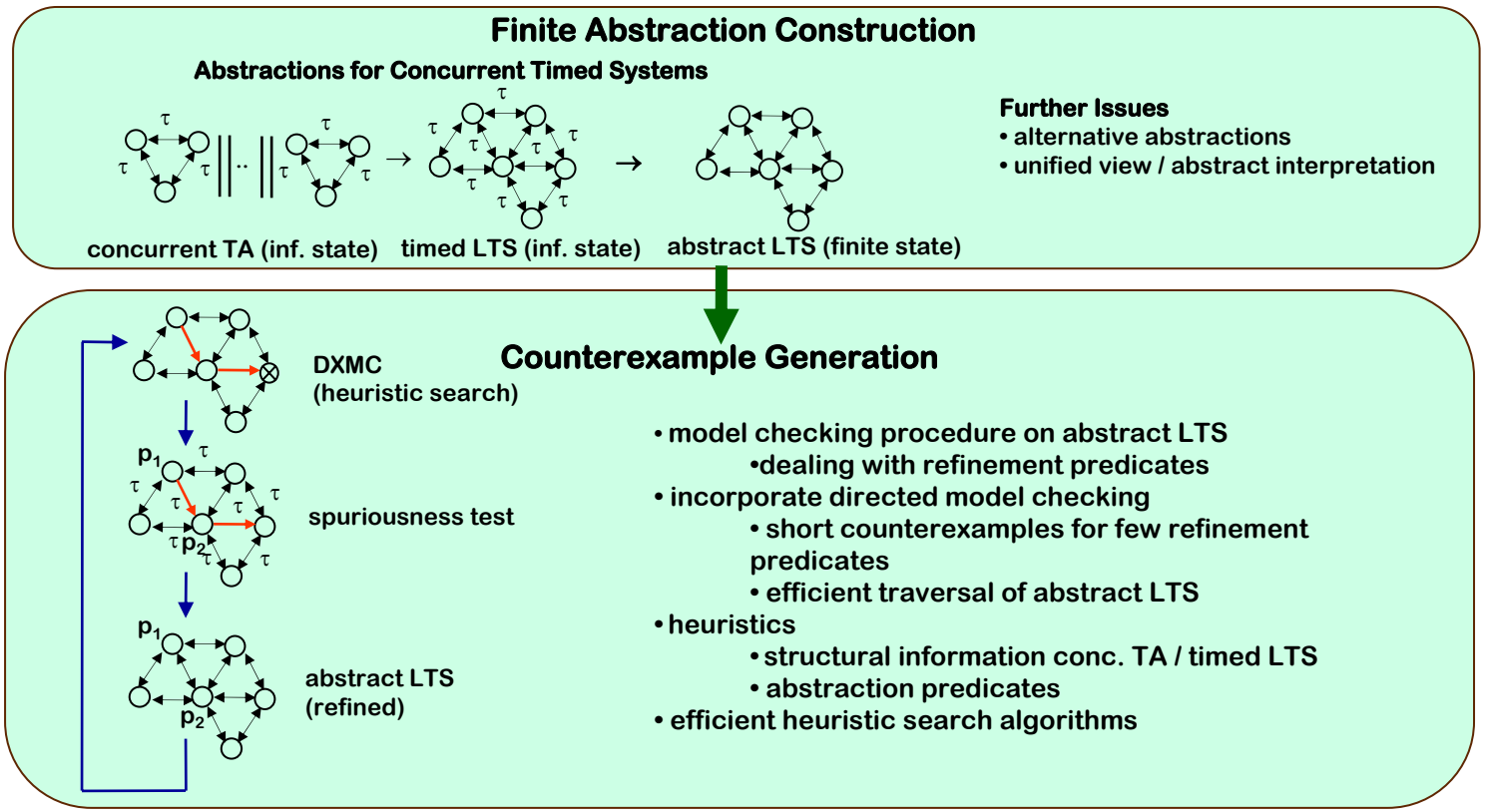
Directed Model Checking on LTS

• short, partial counterexample

  • large amount of probability mass

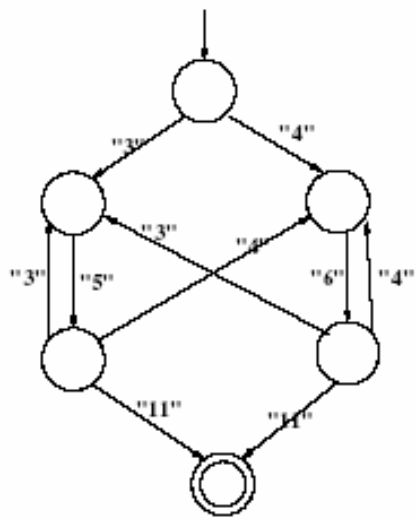
• efficient search in large state spaces

DMC and Abstraction Refinement



Guided Search in FLAVERS

- **Structural Heuristics**
  - e.g., in Java PathFinder (NASA Ames)
    - maximize number of thread switches to detect concurrency related bugs more easily
- **FLAVERS (UMass Amherst)**
  - finite-state verification based on data-flow abstracted model of Ada/Java programs
    - heuristics: minimal distance in task automaton to reach an accepting (= property violating state) [FSE 2004]



Overview

1. Introduction

2. State Space Exploration Strategies

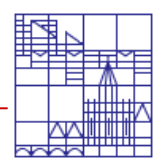
3. Directed Explicit-State Model Checking for Safety Properties

4. Directed Explicit-State Model Checking for Liveness Properties

5. Directed Explicit-State Model Checking and Partial-Order Reduction

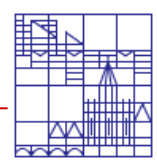
6. Current and Future Research

7. Conclusion



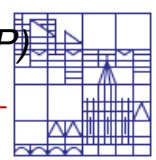
Conclusion

- **Directed Model Checking with A* and BF Searches for Safety Properties**
  - BF for fast error finding, A* for later improvement
  - shorter counterexamples than with DFS, often optimal
  - often less exploration effort than with BFS
  - even crude heuristic estimates improve error trail length
  - for dining philosophers, DMC with A* permits the solution of problems for which DFS fails
- **Liveness Properties**
  - certain improvements through A*+INDFS
  - good results with trail improvement based on Hamming and FSM distance metrics
- **Partial Order Reduction and DMC**
  - approximations $C3_{static}$ and $C3_{duplicate}$
  - occasional loss of optimality
  - good overall co-existence



References

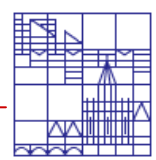
- **[FSE 2004]**
  - J. Tan, G. S. Avrunin, L. A. Clarke, S. Zilberstein and S.Leue, HeuristicGuided, Counterexample Search in FLAVERS, in: Proceedings of ACM SIGSOFT / FSE-12, November 2004.
- **[STTT 2004]**
  - S. Edelkamp, S. Leue, A. Lluch-Lafuente, *Partial Order Reduction and Trail Improvement in Directed Model Checking*, to appear in: Int. Journal on Software Tools for Technology Transfer (STTT), 2004.
- **[SPIN 2002]**
  - A.Lluch Lafuente, S. Edelkamp, and S. Leue, *Partial Order Reduction in Directed Model Checking*, Proceedings of SPIN 2002, LNCS, Vol. 2318, Springer Verlag, April 2002.
- **[STTT 2002]**
  - S. Edelkamp, S. Leue, A. Lluch-Lafuente, *Directed Explicit-State Model Checking in the Validation of Communication Protocols*, to appear in: Int. Journal on Software Tools for Technology Transfer, 2004.
- **[AAAI-SS01]**
  - S. Edelkamp, A. Lluch Lafuente and S. Leue, *Protocol Verification with Heuristic Search*, in: Proc. AAAI-ss01 Workshop on Model-based Validation of Intelligence, Stanford University, March 2001.
- **[SPIN 2001]**
  - S. Edelkamp, A. Lluch Lafuente und S. Leue, *Directed Explicit Model Checking with HSF-SPIN*, in: Proc. of SPIN 2001, LNCS, Vol. 2057, Springer Verlag, May 2001.
- **[SoftMC 2001]**
  - S. Edelkamp, A. Lluch Lafuente und S. Leue, *Trail Directed Model Checking*, in: Proc. Workshop on Software Model Checking, ENTCS, Kluwer, July 2001
- **[STTT 2000]**
  - M. Kamel and S. Leue, *Formalization and Validation of the General Inter-ORB Protocol (GIOP) using Promela and SPIN*, Software Tools for Technology Transfer, April 2000.



HSF-SPIN Experiments

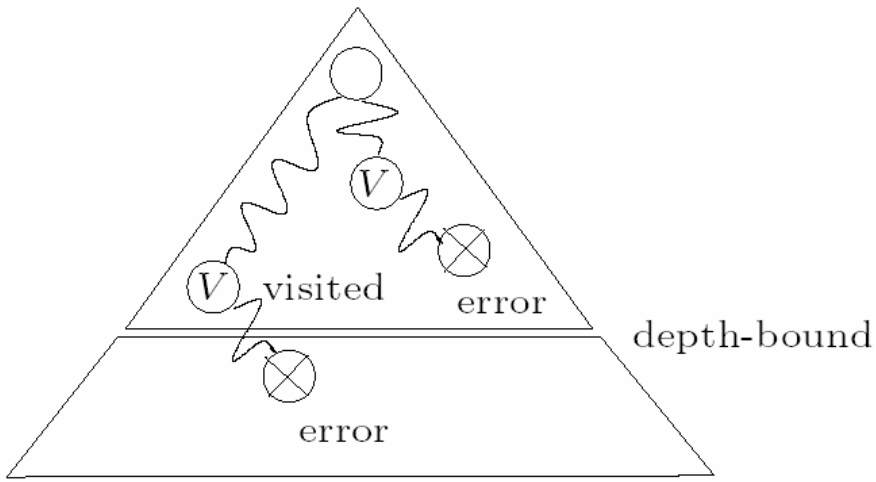
- **Impact of Range of Heuristic Functions**
  - deadlock detection, # of expanded states

Philosophers	no	H_{ap}	H_f	$H_f + U$
e	2,875	17	17	10
r	0..0	0..8	0..10	0..16
Optical	**no**	H_{ap}	H_f	$H_f + U$
e	110,722	14	342	342
r	0..0	0..12	0..14	0..12
Marriers	**no**	H_{ap}	H_f	$H_f + U$
e	493,840	432,483	462,235	192,902
r	0..0	0..4	0..25	0..25
GIOP	**no**	H_{ap}	H_f	$H_f + U$
e	37,266	27,061	28,067	24,859
r	0..0	0..6	0..12	0..25



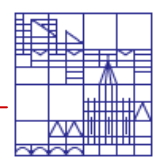
Depth-Bounded DFS

- **Finding Shortest Error Paths**
  - continue to explore when property-violating state found
    - bound search depth to shallowest error found
  - anomaly in depth-bounded search



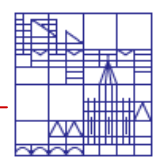
    - the fact that V has already been visited at higher depth means that the shallower V will not be re-explored, i.e., error not found
    - solution: enforce re-exploring of states reached along another, shorter path (-DREACH in SPIN)
      - worst-case exponential time penalty

Graphiken von A. Lluch-Lafuente



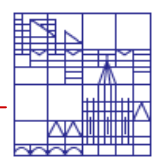
- **Monotonicity of Heuristics**
  - monotonicity
    - for each state u and each successor v of u:

      $h(u) - h(v) \leq cost(u \rightarrow v)$
  - impact
    - non-monotone heuristics may lead to exponential blow-up of reopenings in A*
    - in practice: reopening rarely observed for non-monotone heuristics



IDA* for Safety Properties

```
IDA*(s)
  Push(S, s, h(s)); U ← U' ← h(s)
  while (U' ≠ ∞)
    U ← U'; U' ← ∞
    while (S ≠ ∅)
      (u, f(u)) ← Pop(S)
      if (failure(u)) exit Safety Property Violated
      for all v in Γ(u)
          if (f(u) + 1 − h(u) + h(v) > U)
            if (f(u) + 1 − h(u) + h(v) < U')
              U' ← f(u) + 1 − h(u) + h(v)
          else
            Push(S, v, f(u) + 1 − h(u) + h(v))
```

Complexity Issues

- **Model Checking**
  - Translation LTL -> Büchi automata: exponential in the length of the formula
  - DFS: O(n+e)
- **A***
  - maintaining the nodes
    - open_list: priority queue
      - as heap: O((n+e) log n)
      - as Fibonacci-heaps: O(e + n log n)
  - additional effort for the state exploration (current research)
    - h admissible, h not monotone: exploration of exponentially many nodes
    - h inadmissible (no search for an optimal solution): avoidance of re-opening
      - H_{ap}: monotone, admissible
      - H_D without rendez-vous: monotone, admissible