Directed Model Checking with Distance-Preserving Abstractions

Bernd Finkbeiner
Universität des Saarlandes

joint work with
Klaus Dräger and Andreas Podelski

Model Checking

- **Verification by exhaustive state space exploration**

  Fundamental complexity-theoretic barrier
  State space is exponential in number of components

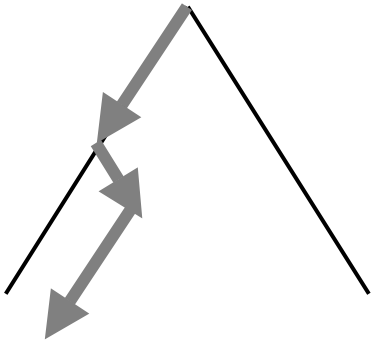
- **Error detection by search**

  Hope despite state space explosion
  A good search strategy is likely to explore only
  a small portion of the state space

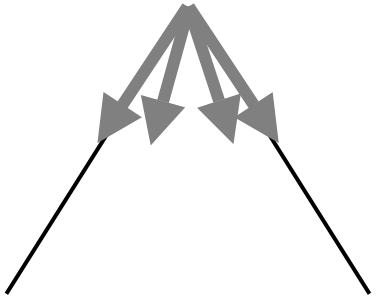
Search Strategies

- **Standard Model Checking**

  Depth-first search

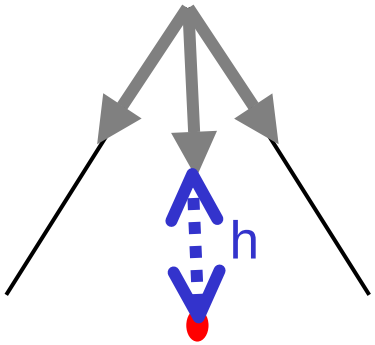


  Breadth-first search

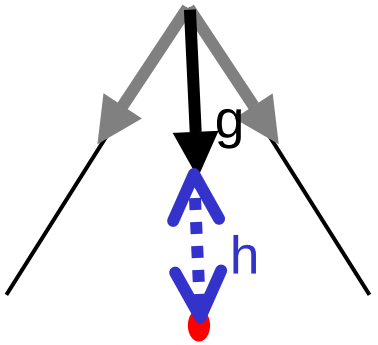


- **Directed Model Checking**

  Best-first search



  A*



Heuristics

- **User-provided hints**
- **Pattern databases**
- **Property-specific heuristics**

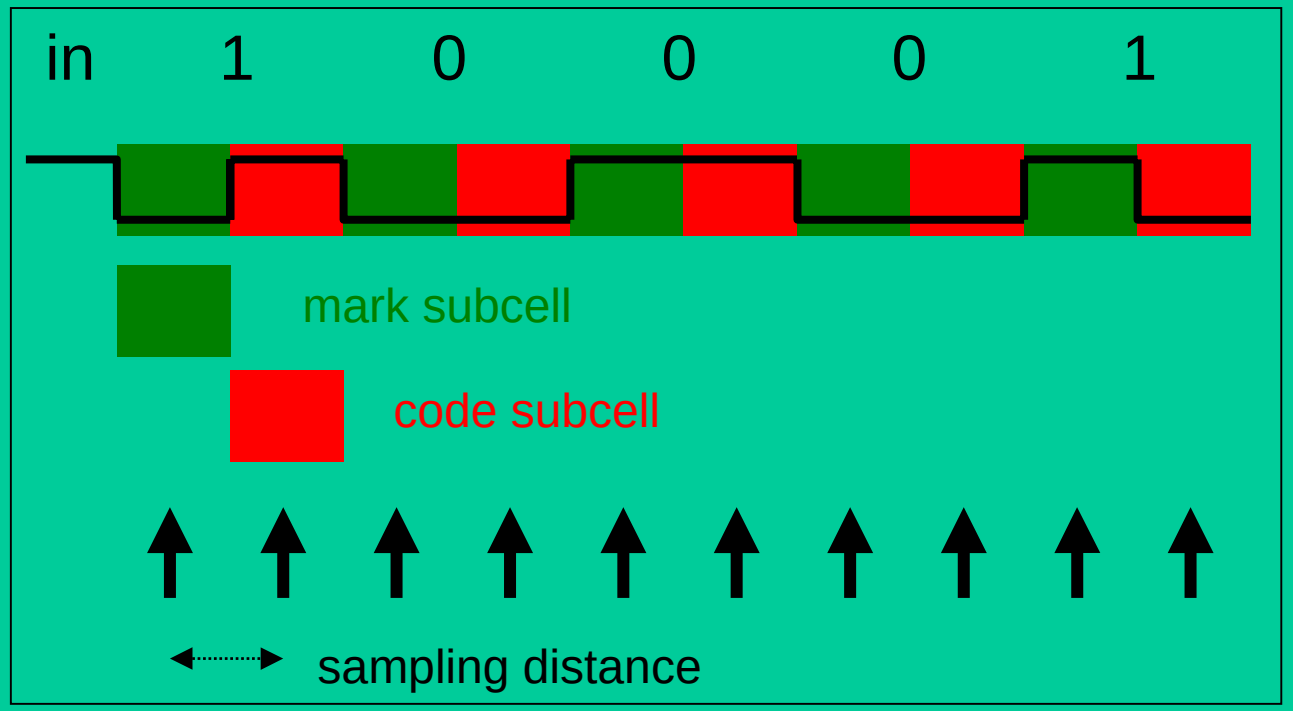
- **Structural heuristics**
- **Hamming Distance**
- **FSM heuristic**

interactive

automatic

User-Provided Hints

Example: Biphase Mark Protocol in UPPAAL



User-provided hint „in=1“: **UPPAAL explores 50% fewer states.**

G. Behrmann, A. Fehnker. Th. Hune, K. Larsen, P. Pettersson, J. Romijn
Efficient Guiding Towards Cost-Optimality in UPPAAL, 2005

Abstraction-based Heuristics

Model checker performs two separate state space traversals:

- Exhaustively traverse the state space of the abstract system
  → generate pattern database

- Selectively traverse the state space of the concrete system
  based on pattern database

Estimate of error distance in concrete system =
Actual error distance in abstract system

Example: Arbitertree

Abstraction by ignoring irrelevant variables (as determined by user)

NuSMV without heuristic: **tree with 15 nodes**

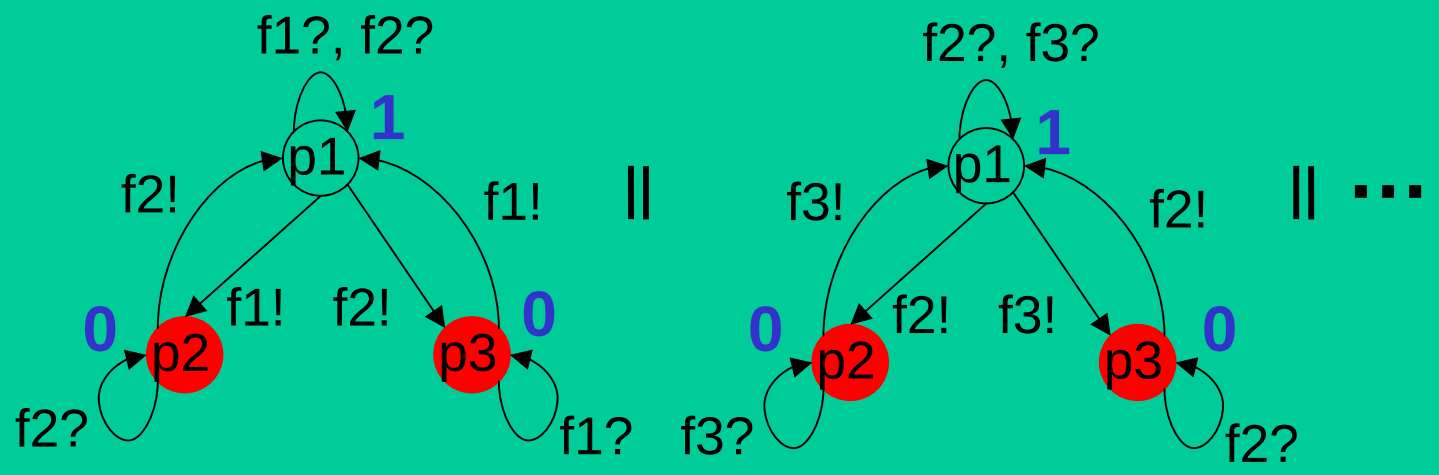
NuSMV with heuristic: **tree with 19 nodes**

K. Quian, A. Nymeyer
Guided Invariant Model Checking Based on Abstraction and Symbolic Pattern Databases, 2004

The FSM Heuristic

Estimate = sum of local error distances

Example: Dining Philosophers



h(p1,p1,p1,p1,p1)=5,
h(p2,p1,p1,p1,p1)=4, …

SPIN without heuristic: **14 philosophers**

HSF-SPIN with heuristic: **254 philosophers**

A. Lluch Lafuente, Directed Search for the Verification of Communication Protocols, 2003

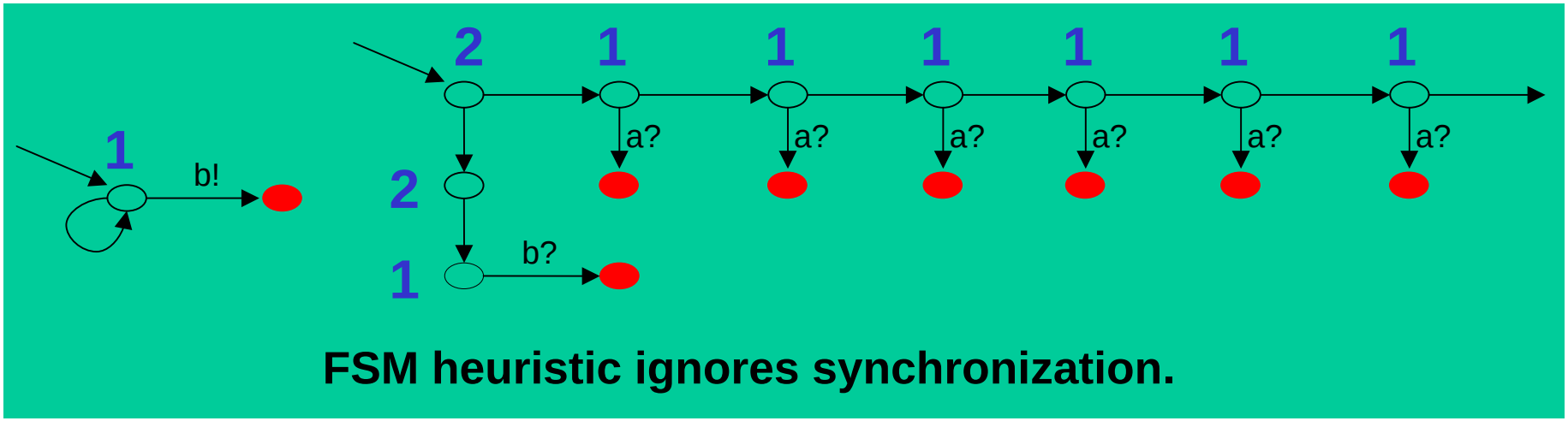
FSM Heuristic applied to Controllers

Explored states (best-first)

Exp	rDFS	FSM
C1	25122	19263
C2	65275	68070
C3	86439	97733
C4	8.5e5	9.8e5
C5	8.3e6	8.8e6
C6	***	**
C7	*	**

Error trace length

Exp	rDFS	FSM
C1	1087	1442
C2	886	2032
C3	786	1663
C4	1680	5419
C5	1900	14163
C6	***	**
C7	*	**



FSM heuristic ignores synchronization.

* timeout; ** out of memory; *** timeout on some instances

Spectrum

- **User-provided estimate functions**
  - **+** highly accurate
  - **-** require user knowledge about system and error

- **Automatic heuristics like FSM**
  - **+** automatic
  - **-** not very accurate

Can we automatically compute a good estimate?

Trade-Off

- **If the estimate is to be very accurate, then its computation is expensive**

  state space explosion
  during computation of the heuristic

- **If the estimate is very inaccurate, then the search based on the estimate is expensive**

  state space explosion
  during search

Is there a sweet spot in this trade-off?

Approach

- **Pattern Database obtained by abstraction**

  We construct the abstract state space by merging states into equivalence classes.

- **Incremental bounded abstraction**

  We fix a bound *N* on the abstract state space.
  Every composition of two components is followed by an abstraction step that ensures that the size stays below the bound.

Approach can be scaled
to any any desired degree of accuracy.

Performance

- **Best-first search**

	explored states				seconds				trace length			
Exp	rDFS	FSM	N50	N100	rDFS	FSM	N50	N100	rDFS	FSM	N50	N100
C1	25122	19263	871	810	0.24	0.24	0.30	0.49	1087	1442	188	191
C2	65275	68070	1600	2620	0.56	0.59	0.40	1.03	886	2032	203	206
C3	86439	97733	2481	2760	0.74	0.82	0.47	1.14	786	1663	204	198
C4	8.5e5	9.8e5	22223	25206	6.52	6.90	0.91	1.83	1680	5419	247	297
C5	8.3e6	8.8e6	1.6e5	1.6e5	66.41	66.85	2.90	3.97	1900	14163	322	350
C6	***	**	1.7e6	1.2e6	1181	**	18.32	14.87	***	**	480	404
C7	*	**	1.3e7	1.3e7	*	**	156.1	162.4	*	**	913	672
C8	*	**	1.4e7	1.2e7	*	**	163.0	155.3	*	**	1305	2210
C9	*	**	**	3.6e7	*	**	**	1046	*	**	**	1020

* timeout; ** out of memory; *** timeout on some instances

- **A***

	explored states			seconds			trace
Exp	FSM	N50	N100	FSM	N50	N100	length
C1	35768	13863	13455	0.37	0.42	0.62	55
C2	110593	38483	36888	0.99	0.76	1.37	55
C3	144199	44730	42366	1.27	0.91	1.54	55
C4	1.4e6	368813	354091	11.23	4.30	5.05	56
C5	1.3e7	2.8e6	2.7e6	116.28	29.60	29.97	57
C6	*	2.8e7	2.7e7	*	377.77	364.15	57

* (**) out of memory (on some instances)

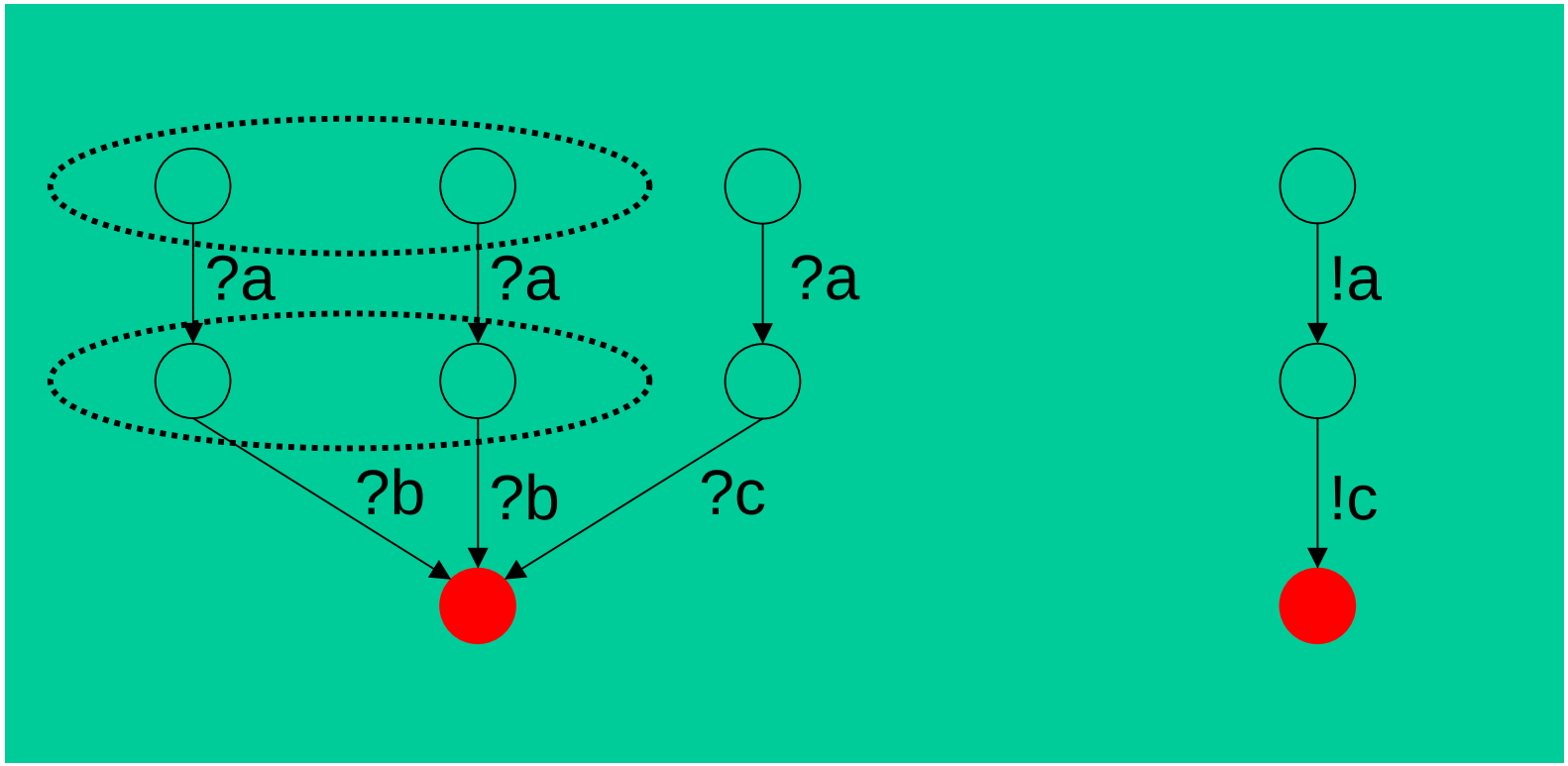
Parameters

- **Abstraction**
  Which states should be merged?

- **Composition strategy**
  In which order should the components be composed?

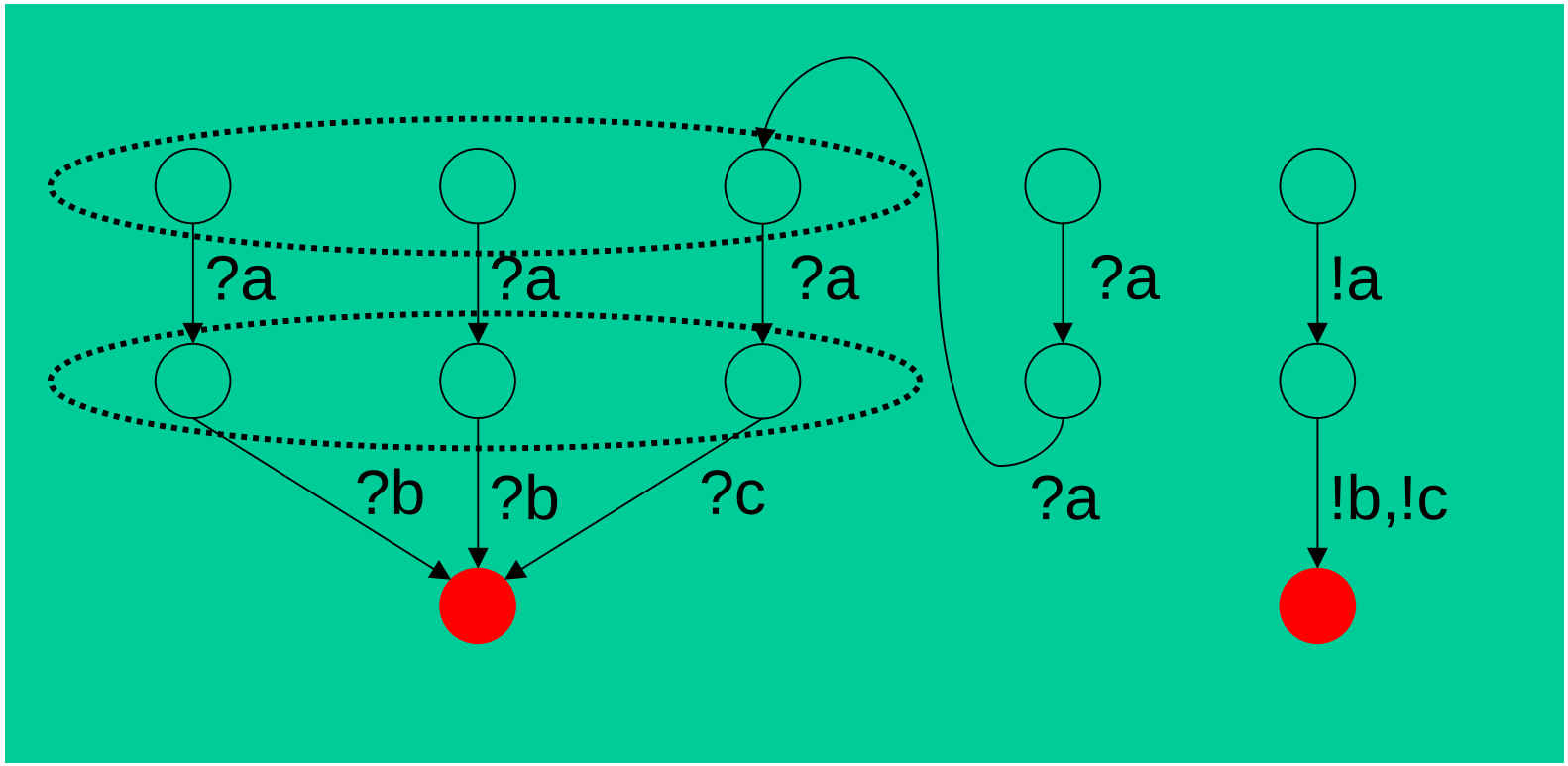
- **Abstraction bound *N***
  How large may the state space grow before states need to be merged?

Language-Preserving Abstraction



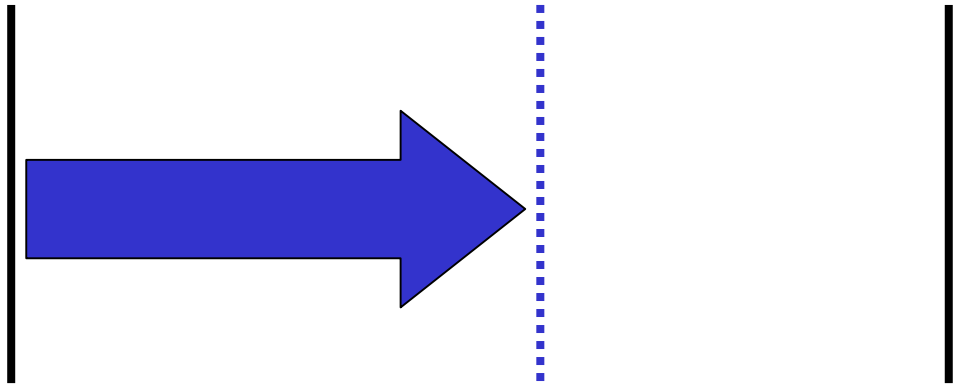
Ideally, the abstraction would be language-preserving.

Distance-Preserving Abstraction



At least, the abstraction should be distance-preserving.

Bounded Abstraction



Distance-preserving abstraction

+ linear cost in # processes

- ignores synchronization

Language-preserving abstraction

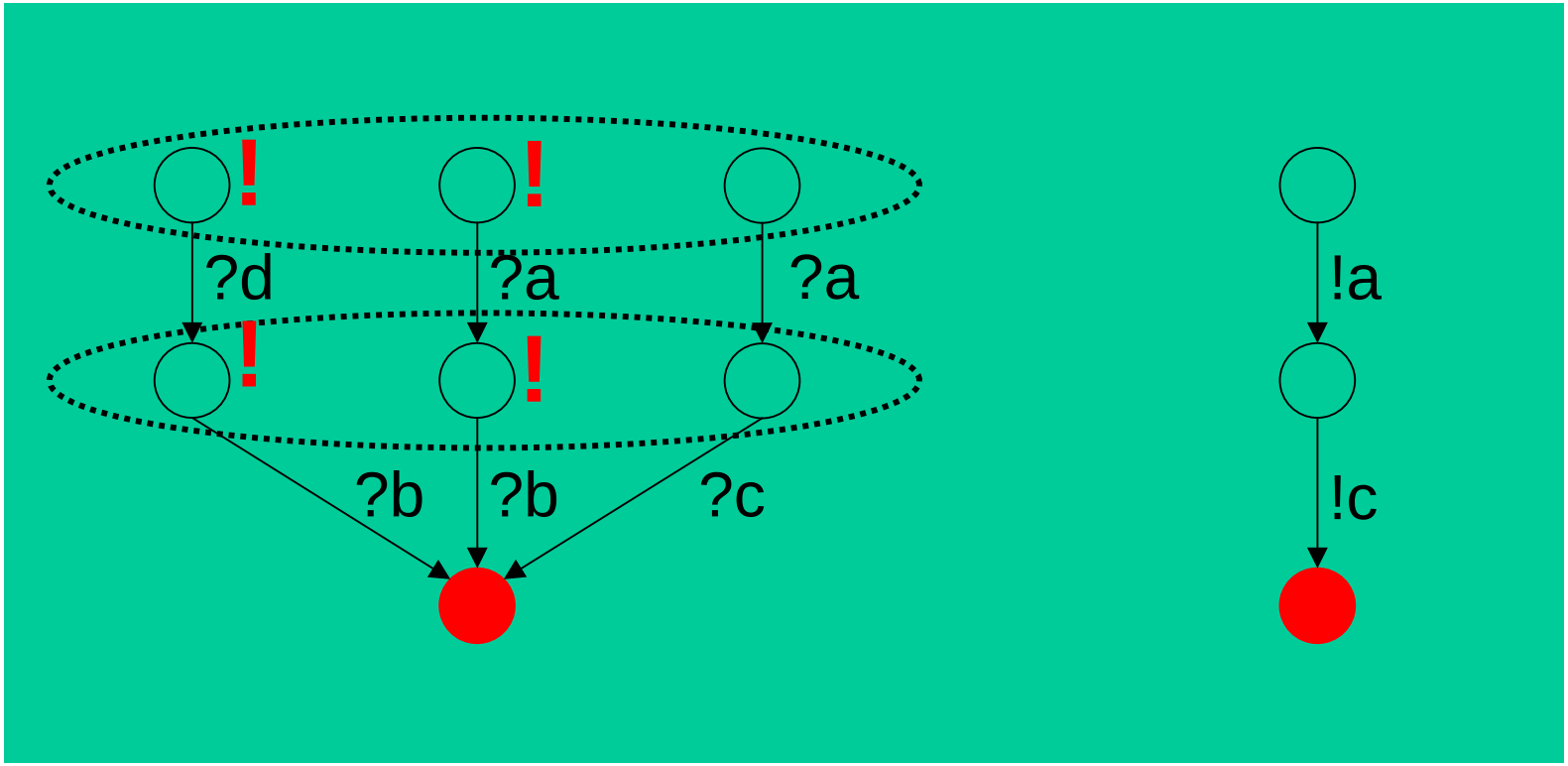
- exponential cost in # processes

+ 100% accurate

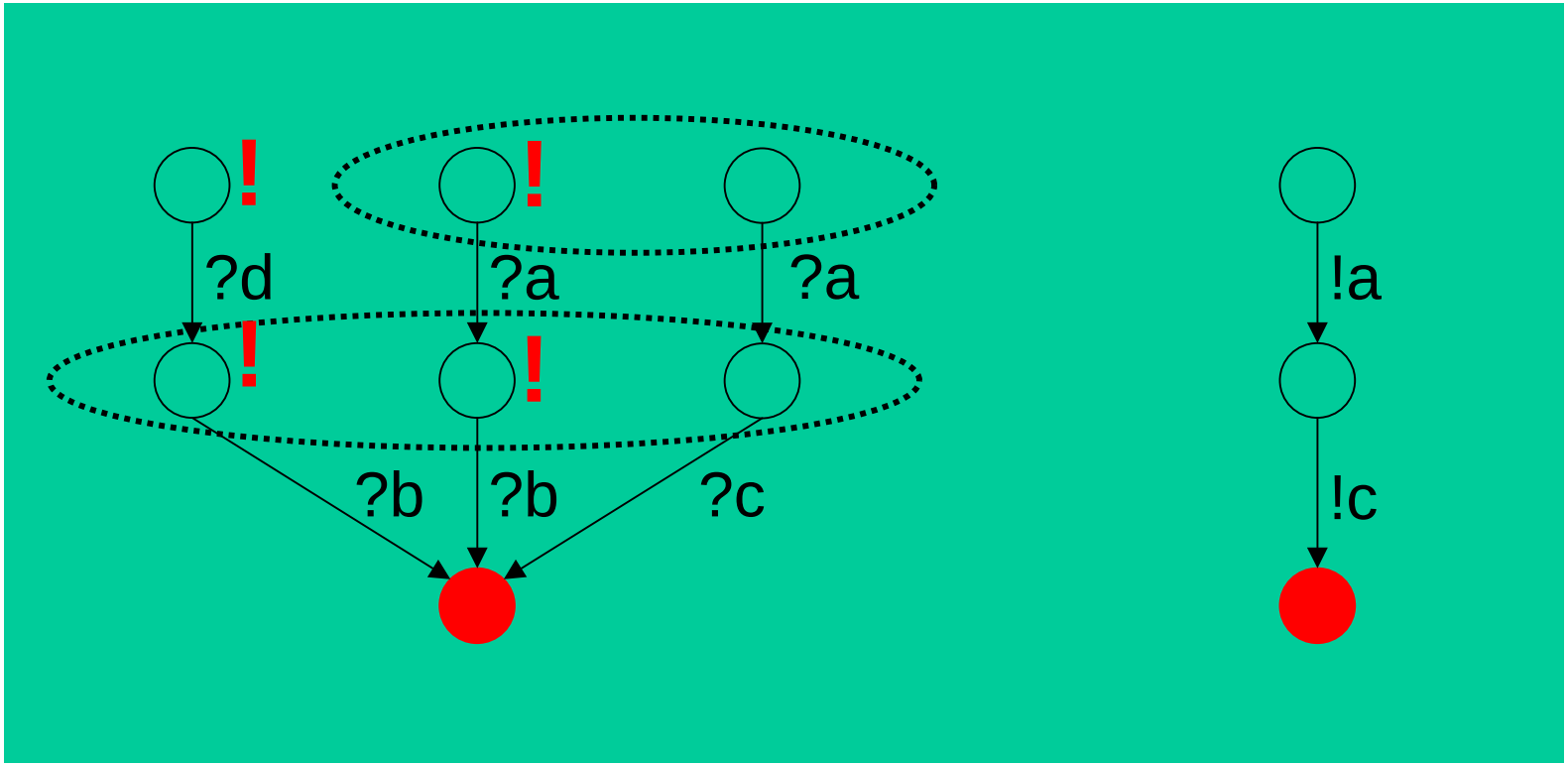
We refine the abstraction until the fixed bound *N* is reached.

linear cost in # processes, accuracy depends on *N*

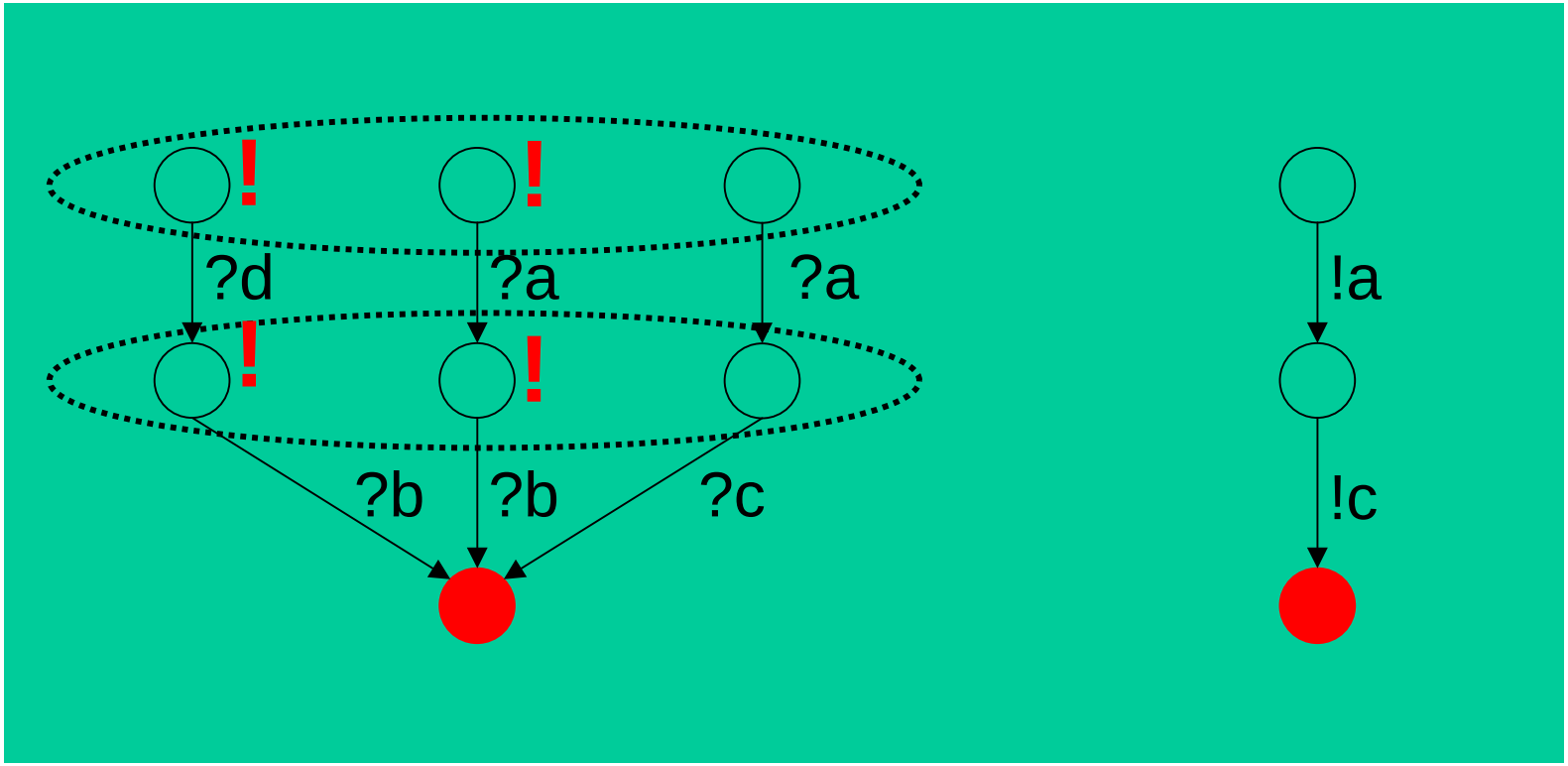
Refining the Abstraction



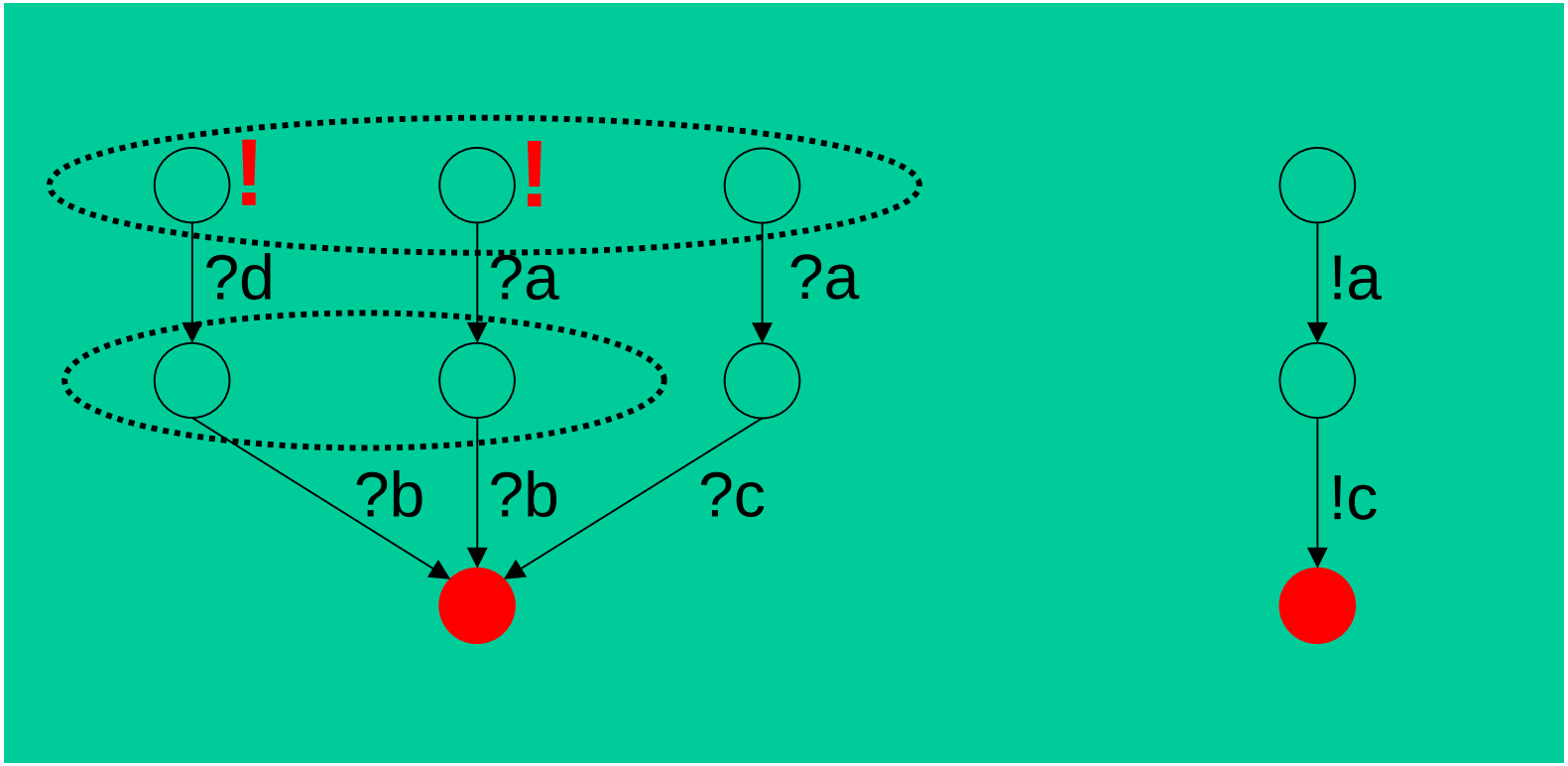
Refining abstraction for states with high error distance



Refining the Abstraction

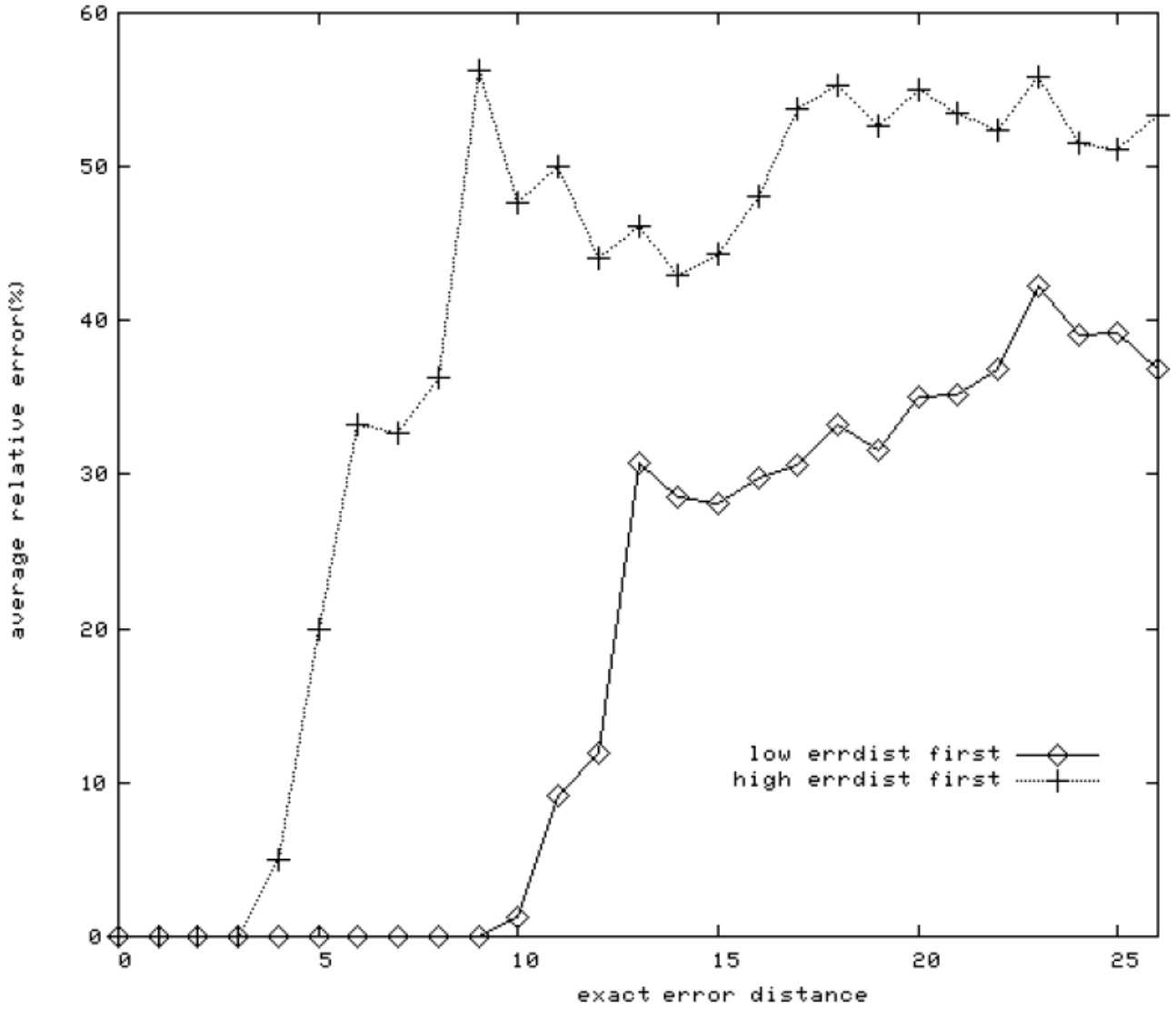


Refining abstraction for states with low error distance



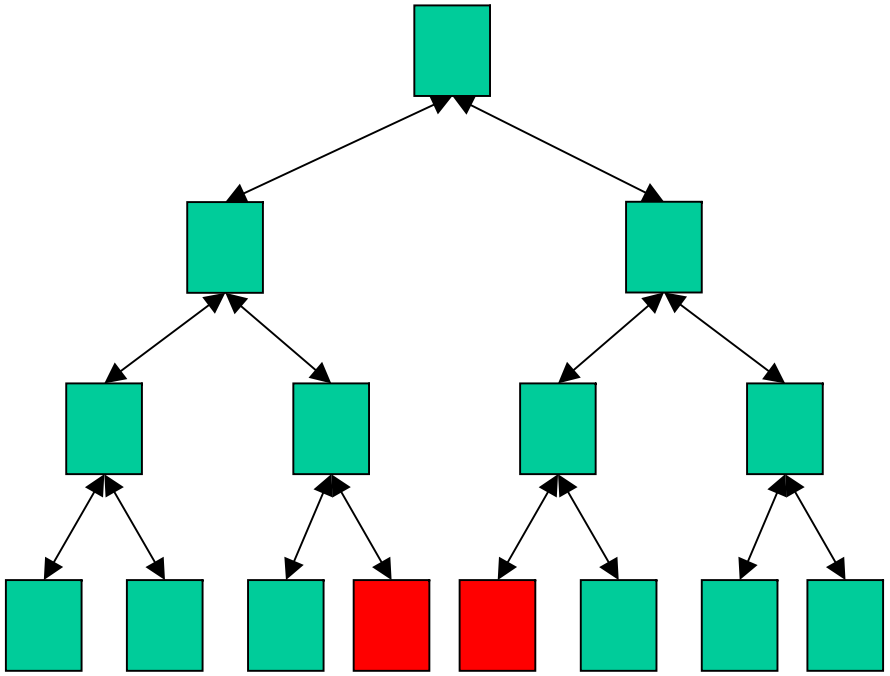
Refinement is most helpful for states with low error distance.

Parameter 1: Abstraction



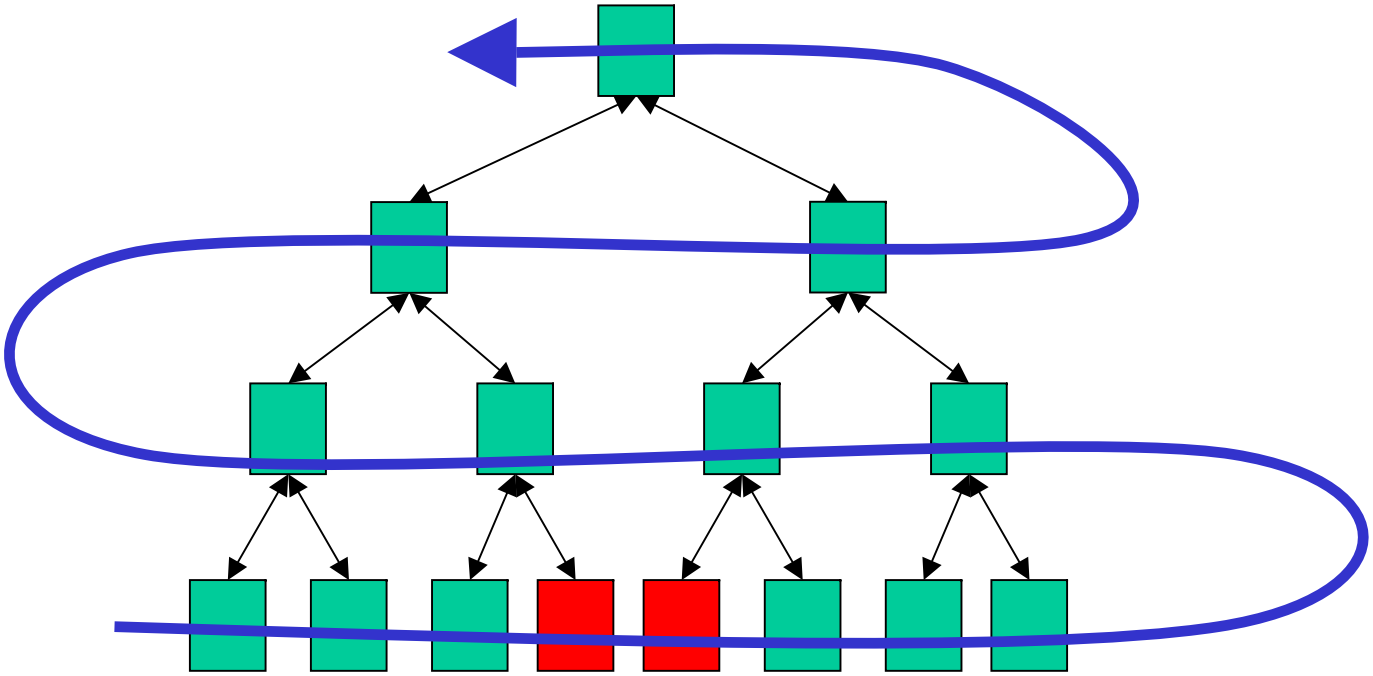
Data based on model checking runs on the „Towers of Hanoi“ benchmark.

Parameter 2: Composition Strategy



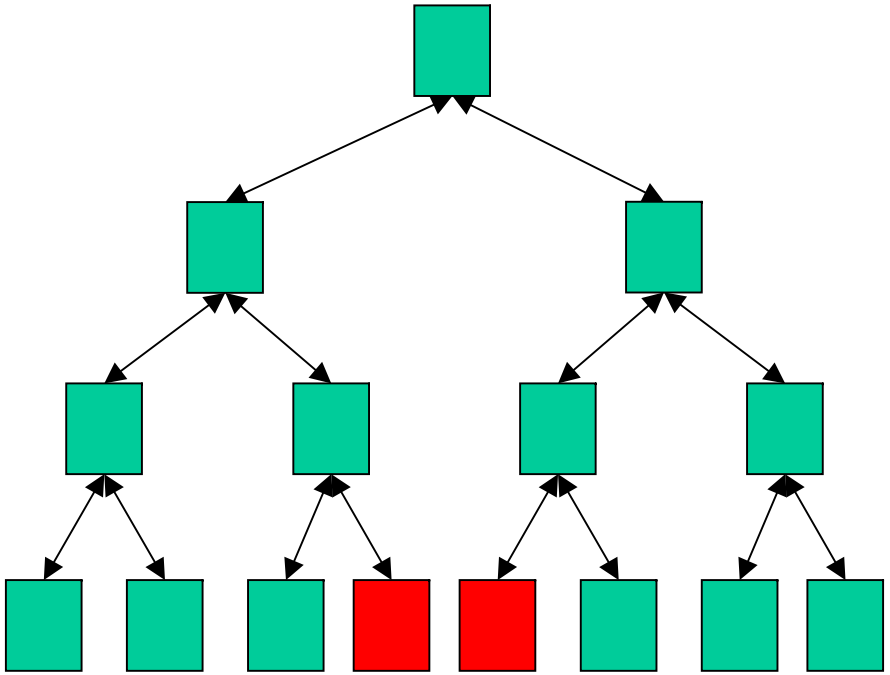
Composition strategy determines the order in which the components are composed.

Default Composition Strategy



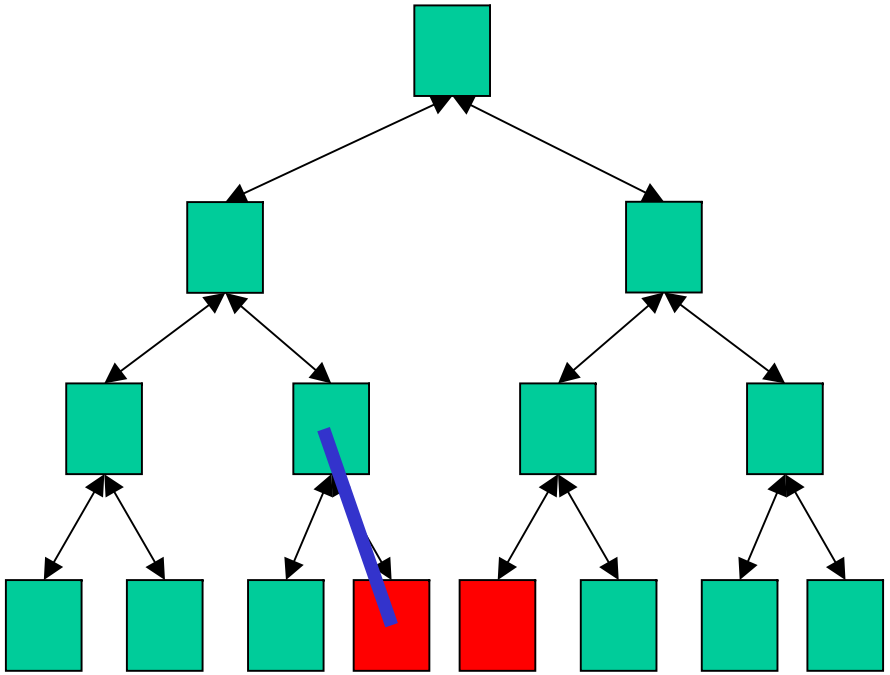
Default Composition strategy composes components in syntactic order

Ranking-based Composition Strategy



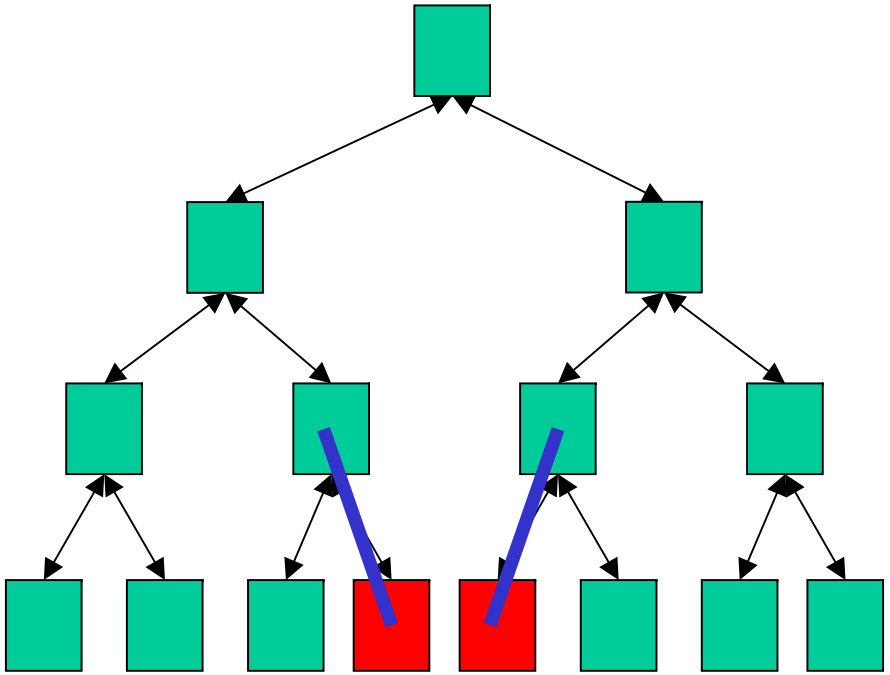
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Ranking-based Composition Strategy



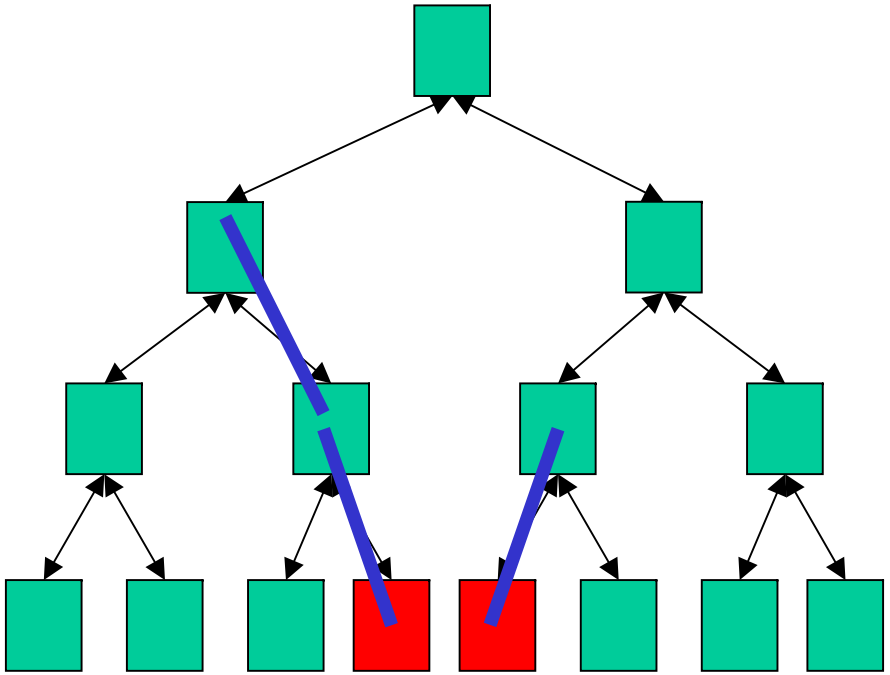
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Ranking-based Composition Strategy



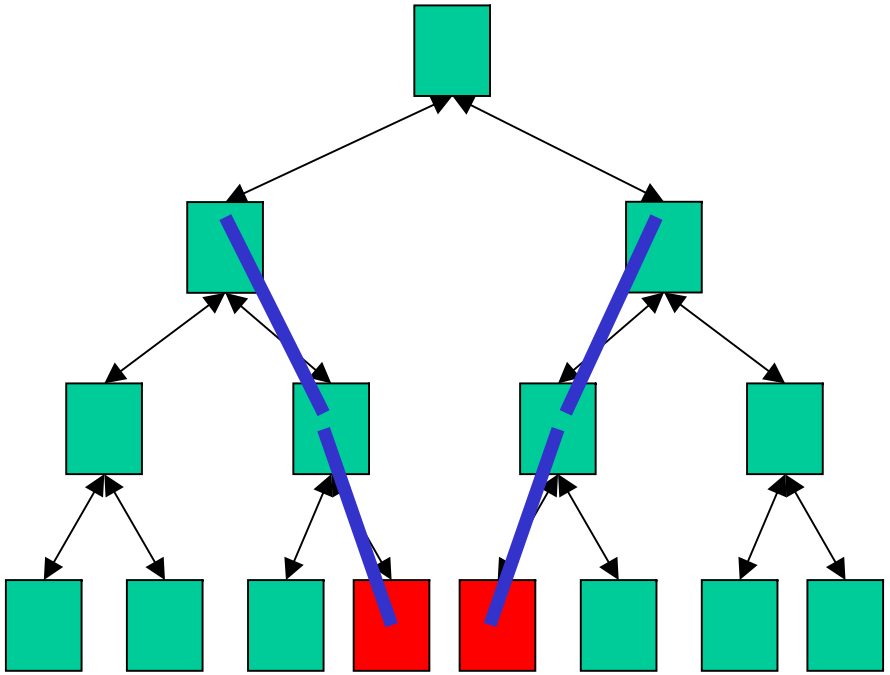
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Ranking-based Composition Strategy



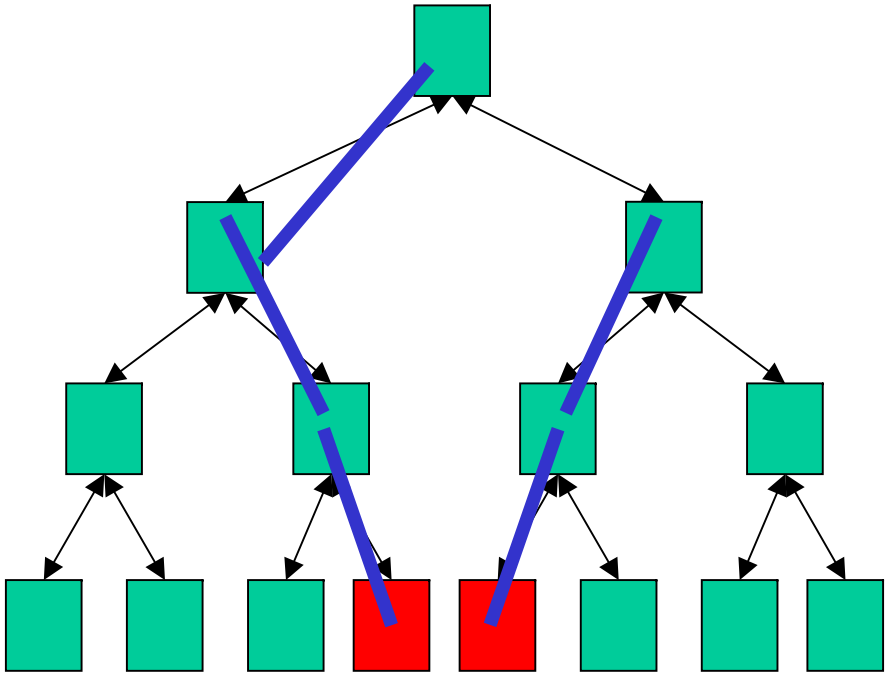
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Ranking-based Composition Strategy



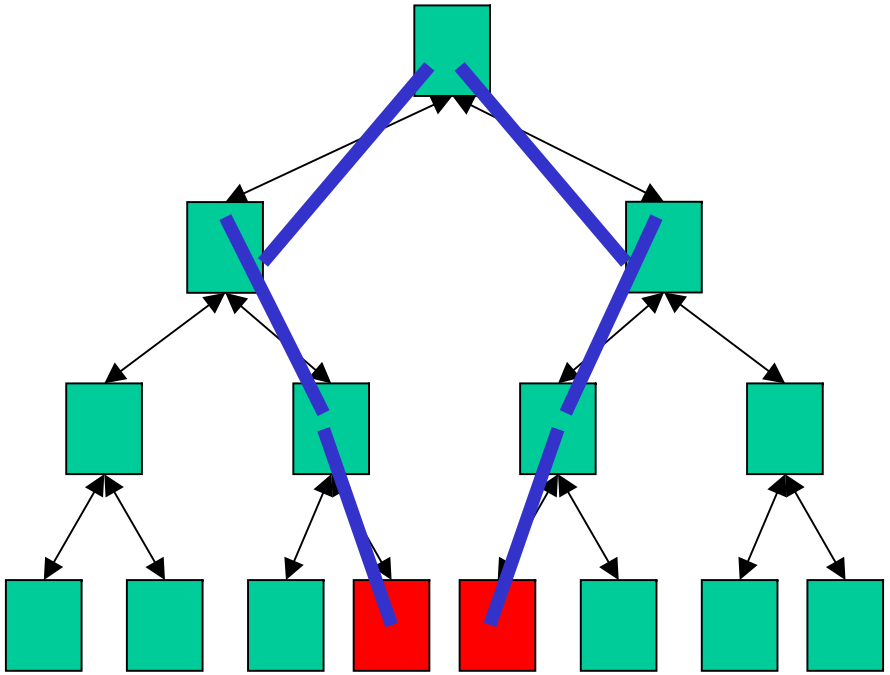
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Ranking-based Composition Strategy



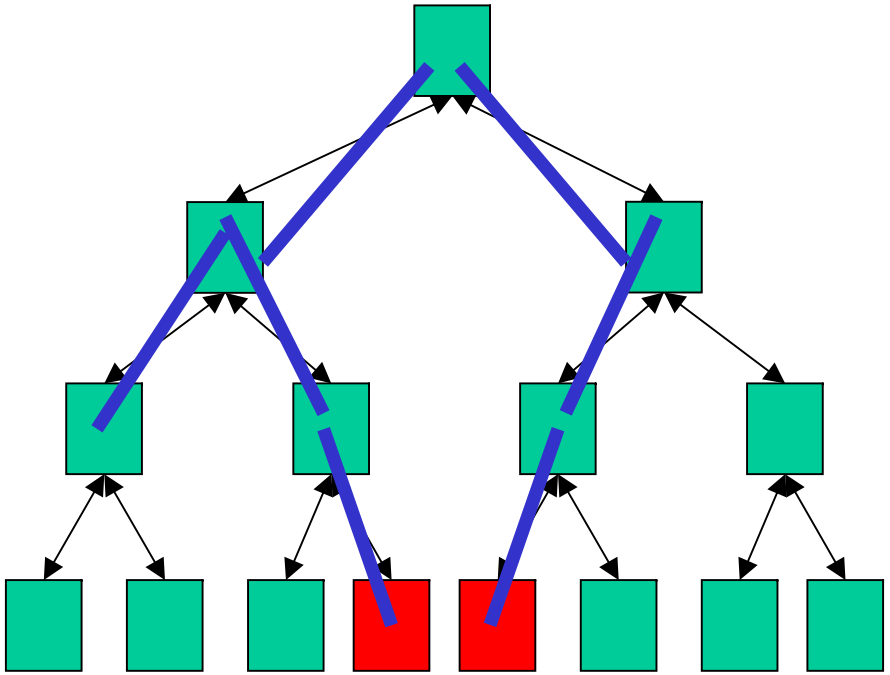
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Ranking-based Composition Strategy



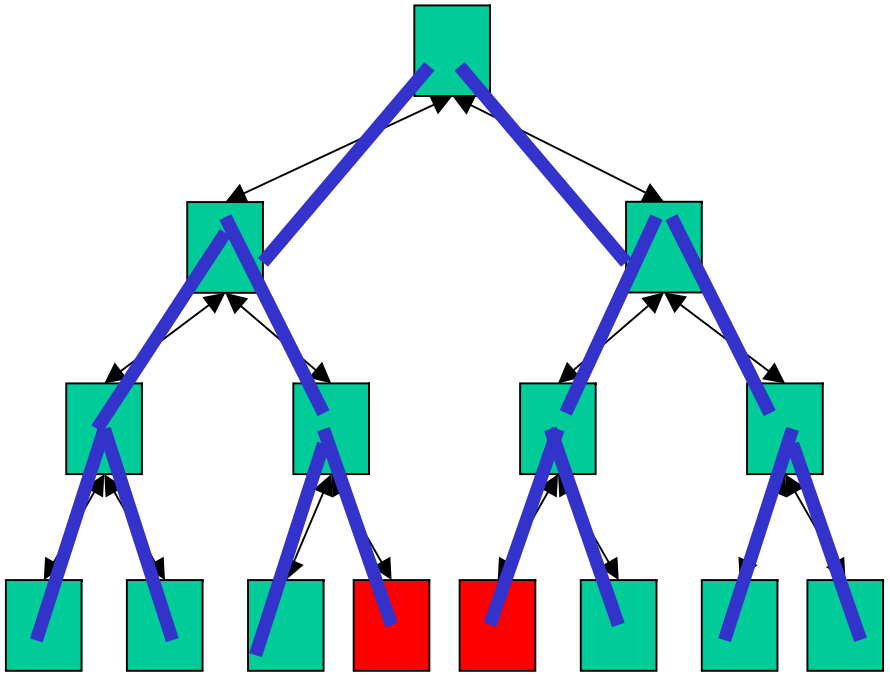
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Ranking-based Composition Strategy



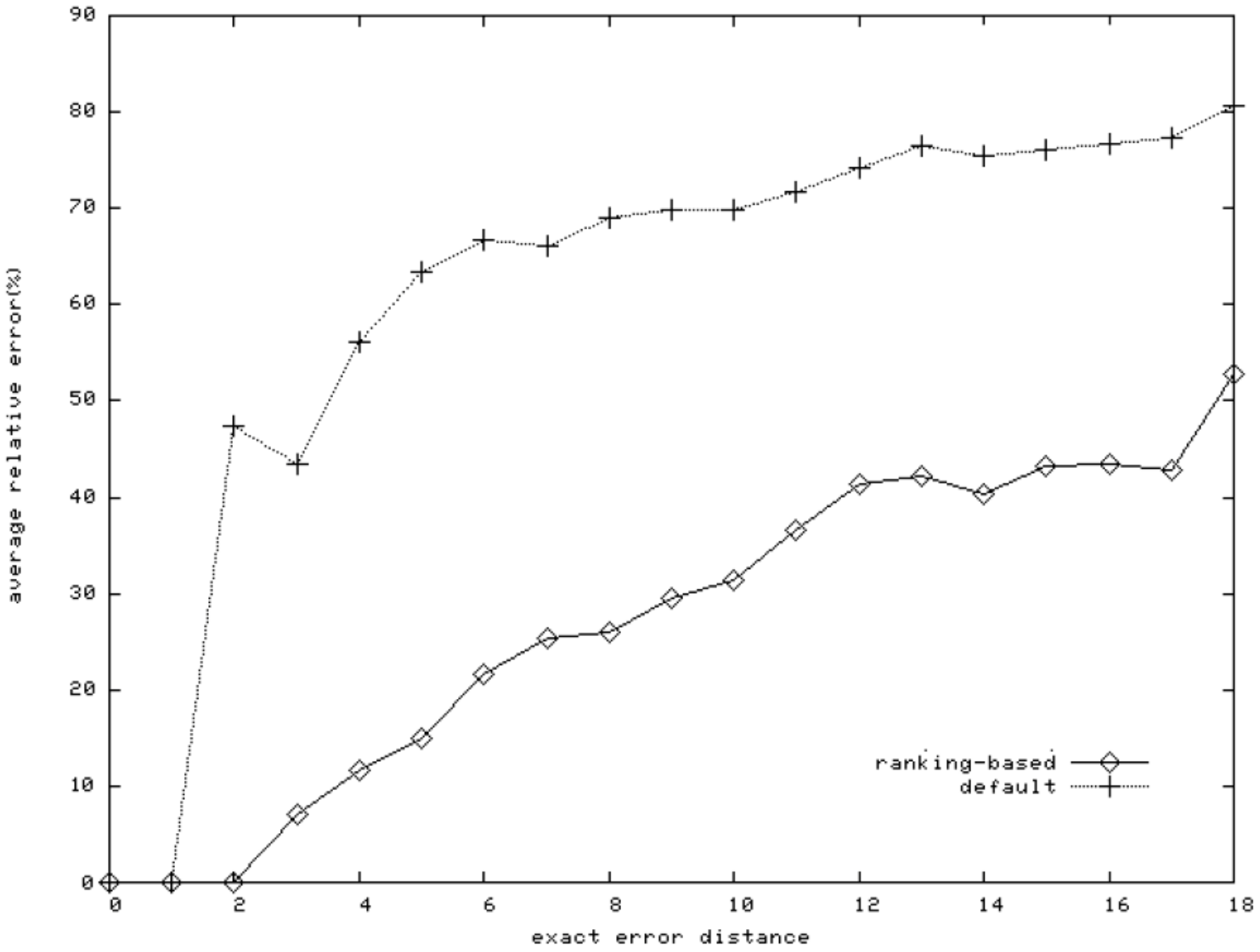
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Ranking-based Composition Strategy



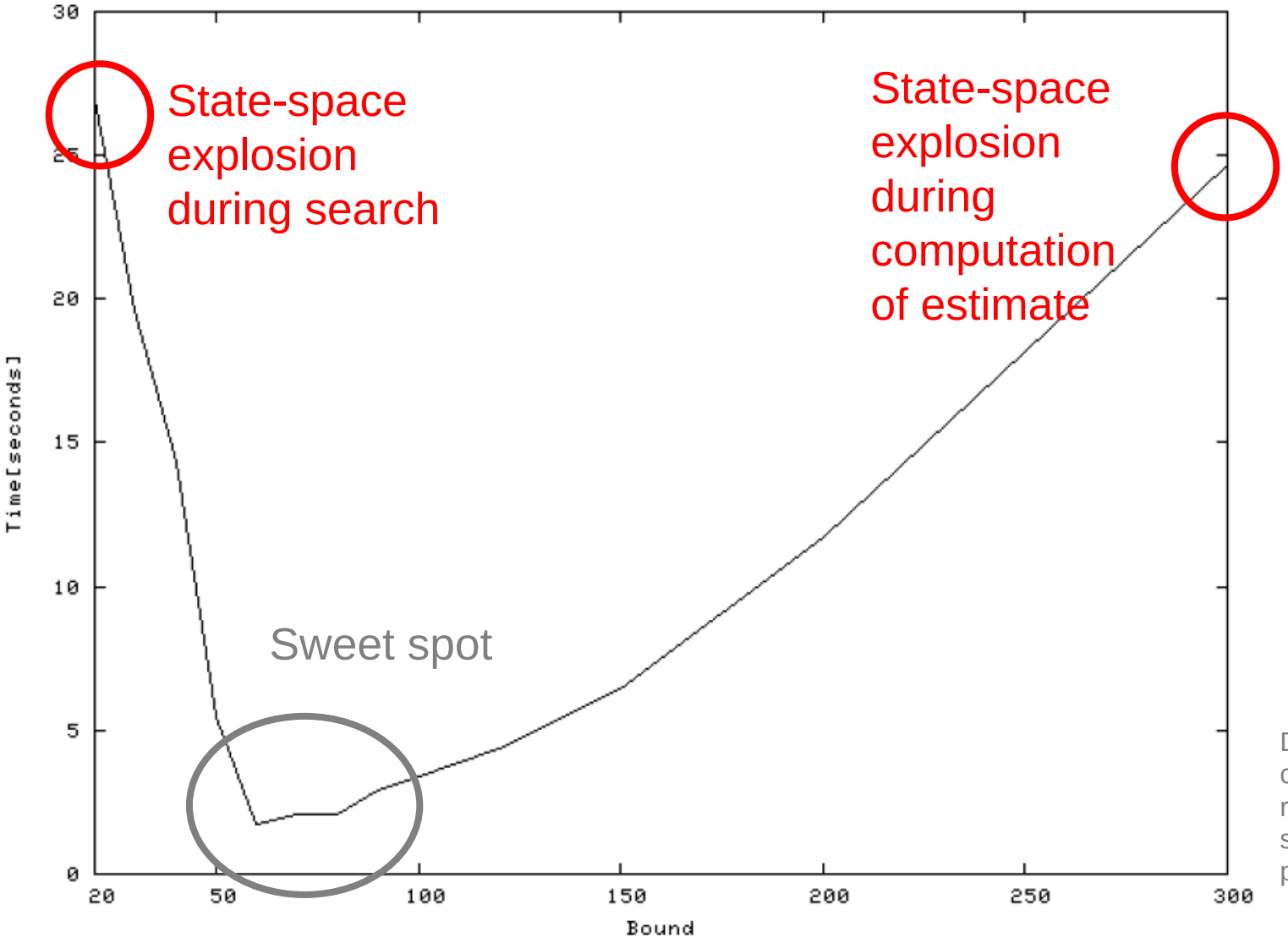
Ranking-based Composition strategy composes components according to synchronization in the proximity of the error

Parameter 2: Composition Strategy



Data based on model checking runs on the „Arbiter Tree“ benchmark with 8 processes.

Parameter 3: Abstraction Bound



Data based on model checking runs with randomly generated systems with 8 processes

Conclusions

- Directed model checking can dramatically speed up model checking
- There is a trade-off between abstraction and search
- Simple heuristics like FSM represent one choice, but not necessarily a good one
- The trade-off appears to have a sweet spot

- Open question:
  How can we systematically find the sweet spot?