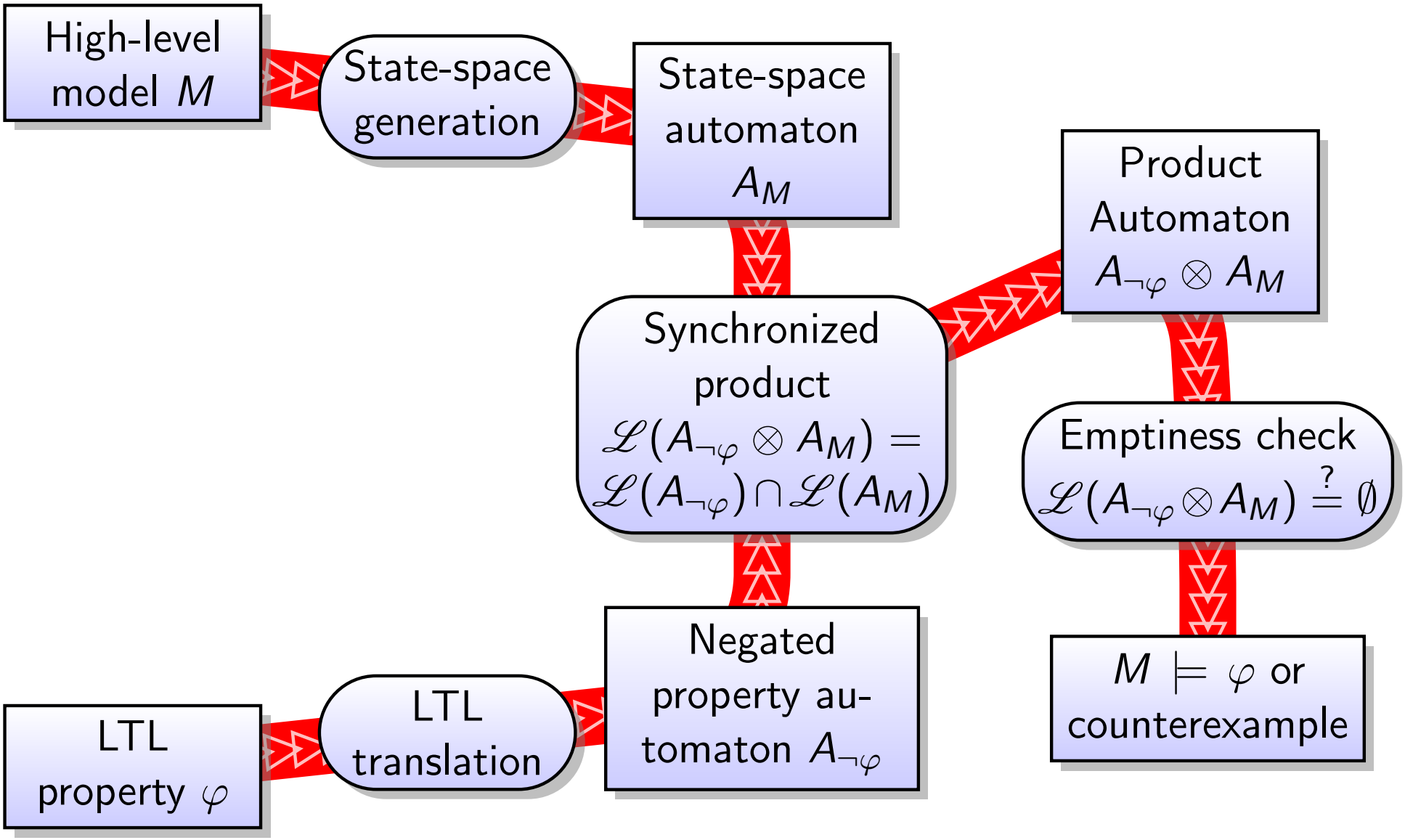
Self-Loop Aggregation Product — A New Hybrid Approach to On-the-Fly LTL Model Checking

Alexandre Duret-Lutz (LRDE/EPITA)
Kais Klai (LIPN/Paris 13)
Denis Poitrenaud (LIP6/Paris 6)
Yann Thierry-Mieg (LIP6/Paris 6)

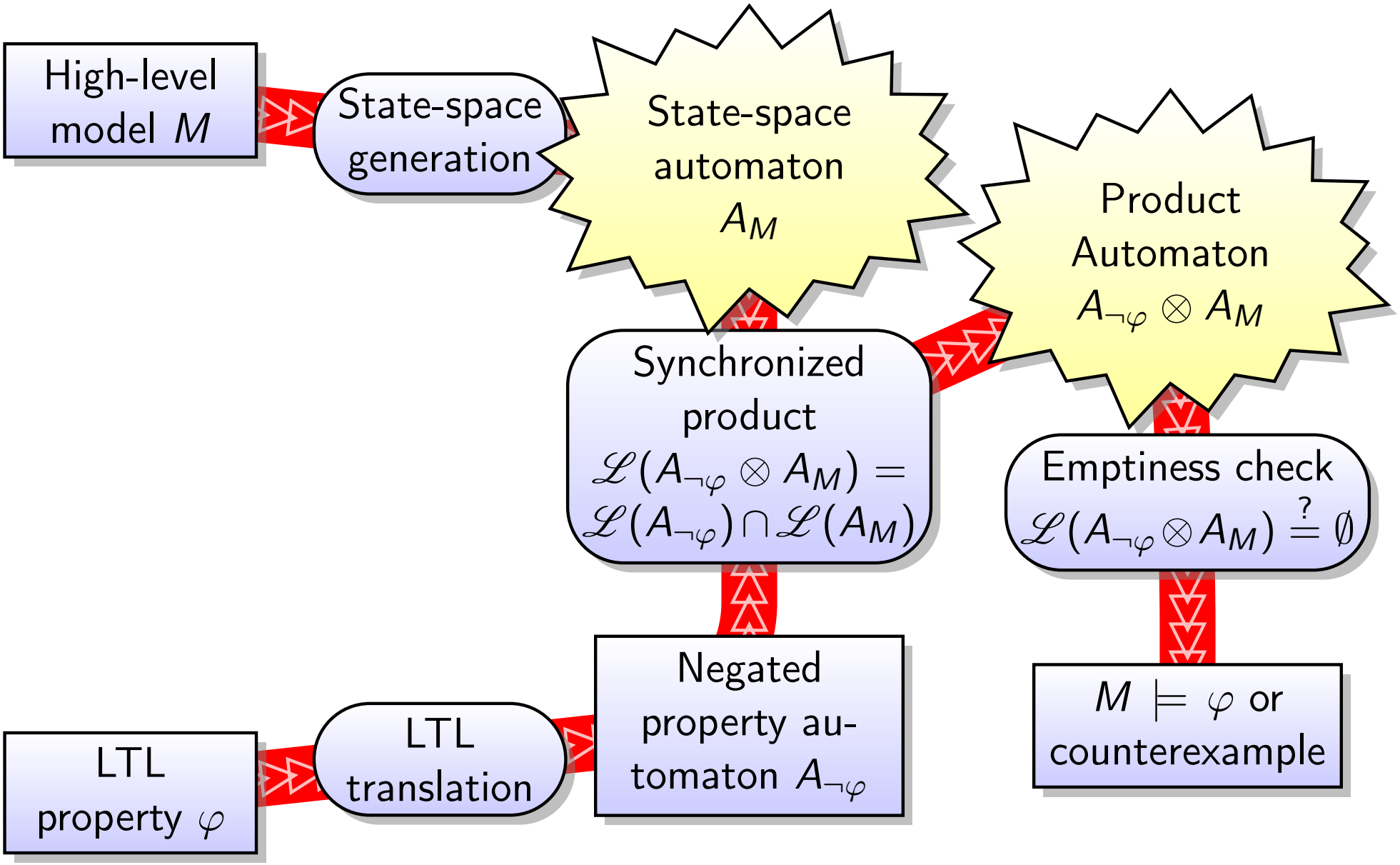
ATVA 2011
October 2011

`http://move.lip6.fr/software/DDD/ltl_bench.html`

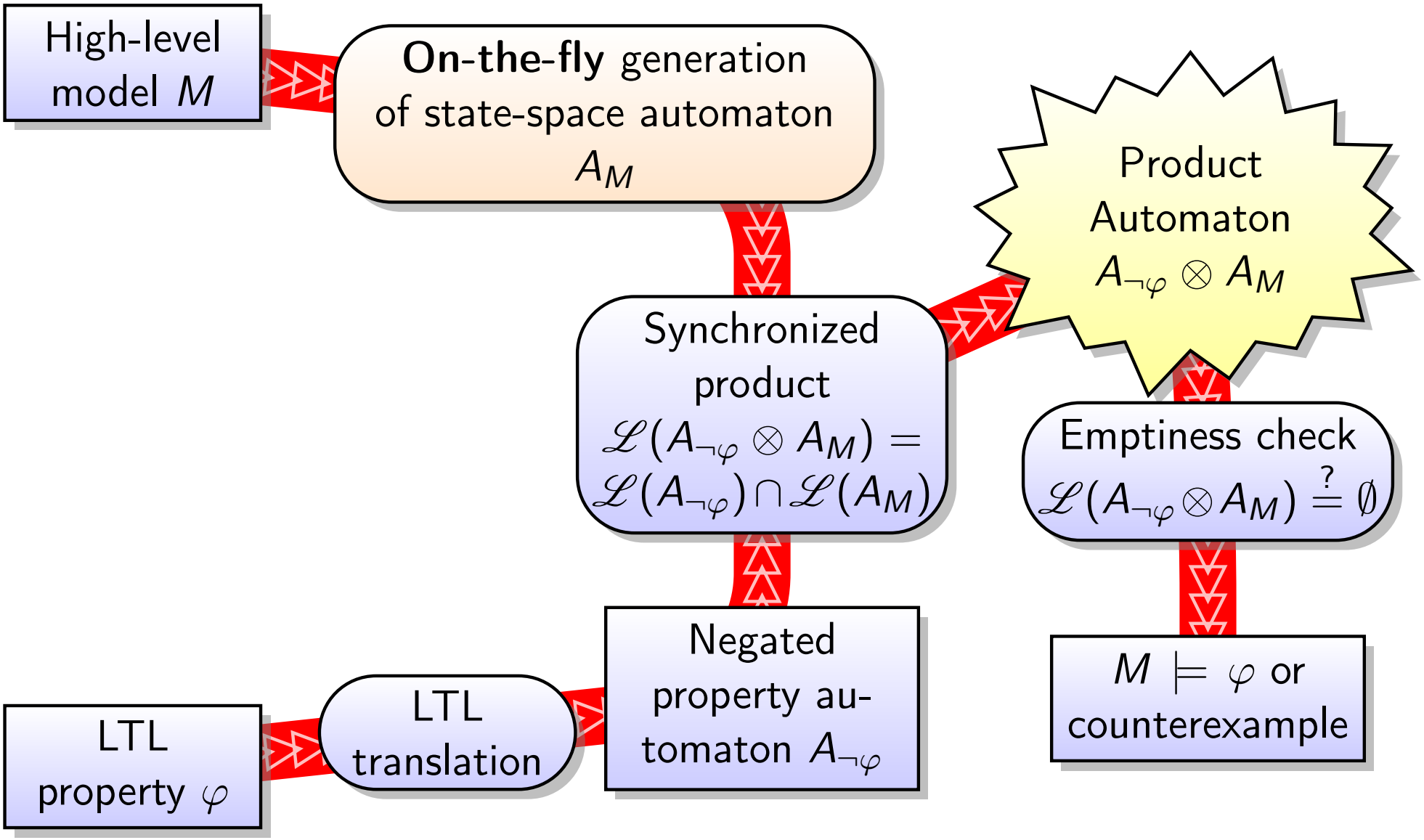
Automata-Theoretic Explicit LTL Model Checking



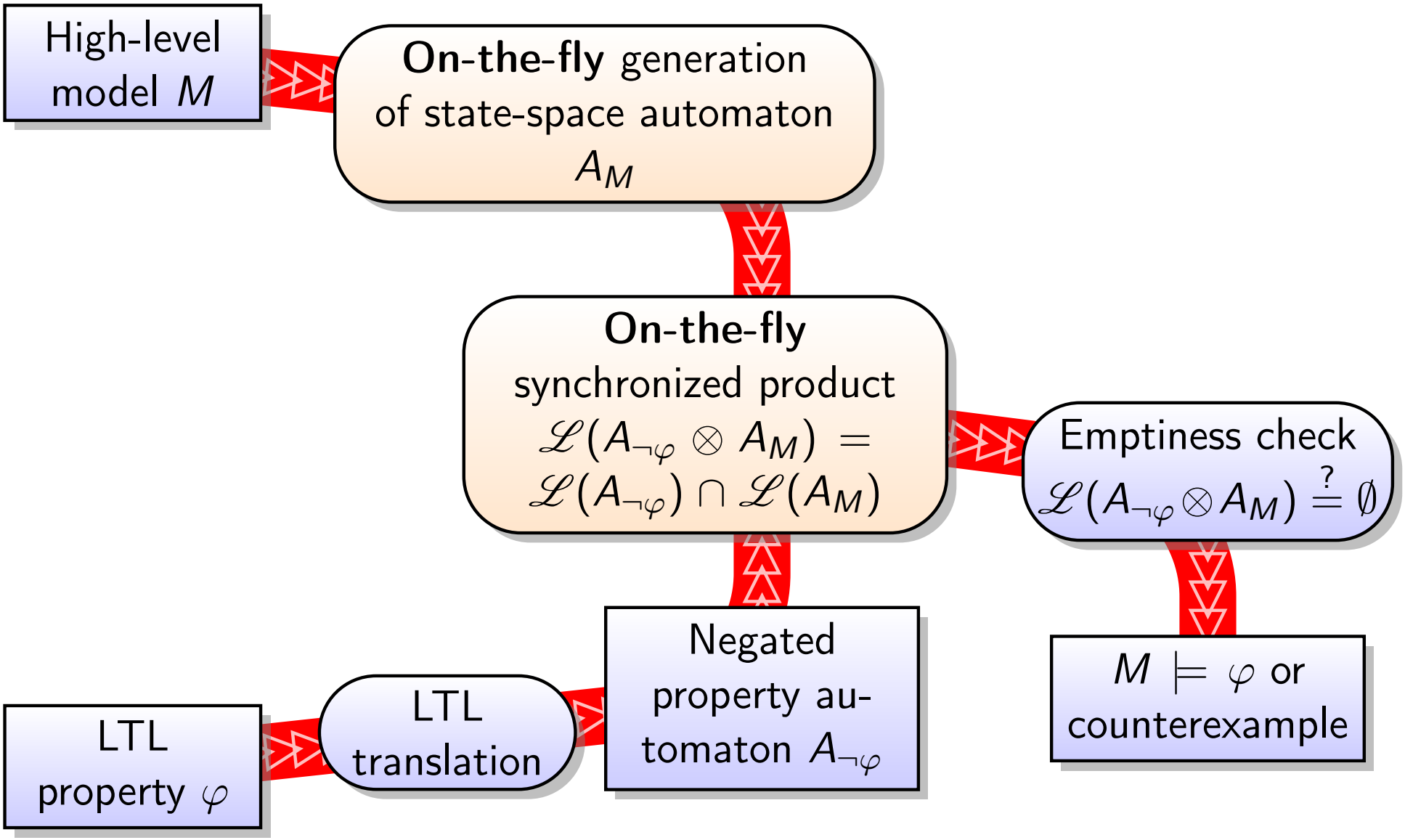
Automata-Theoretic Explicit LTL Model Checking



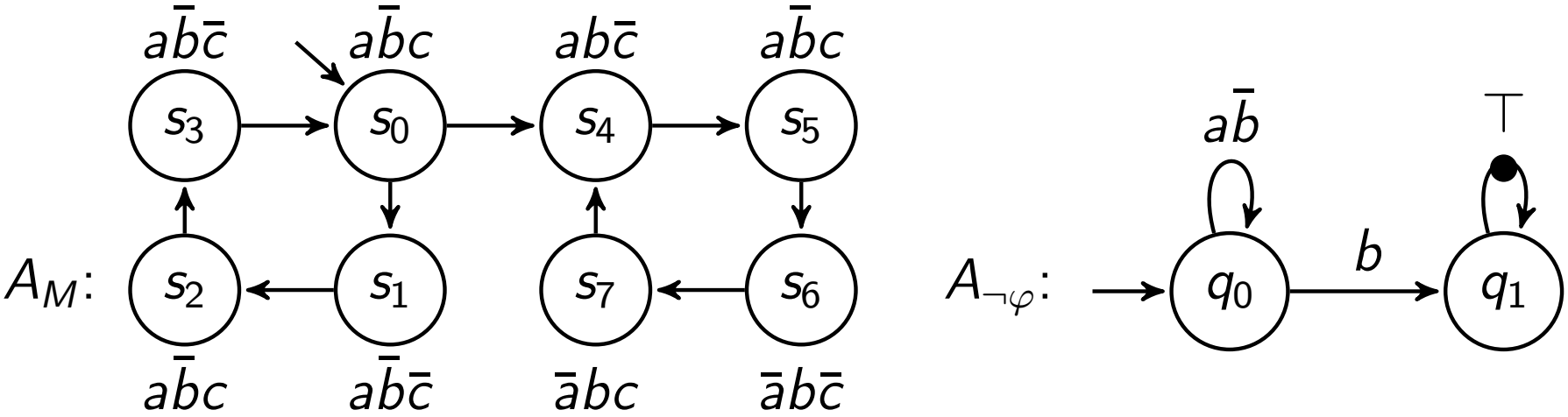
Automata-Theoretic Explicit LTL Model Checking



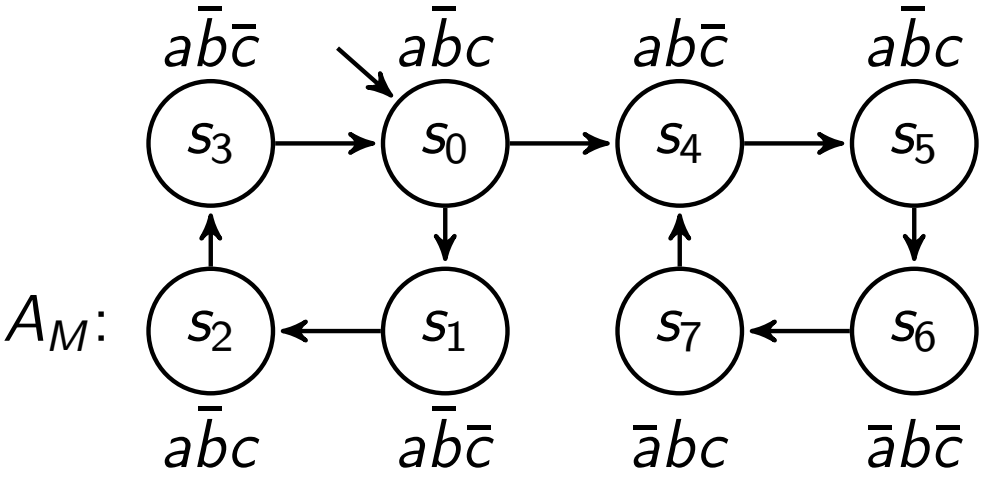
Automata-Theoretic Explicit LTL Model Checking

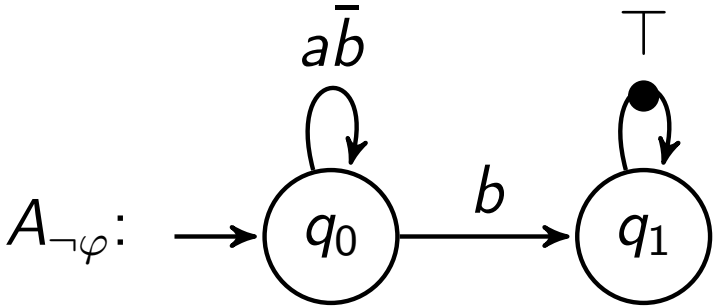


Explicit Approach

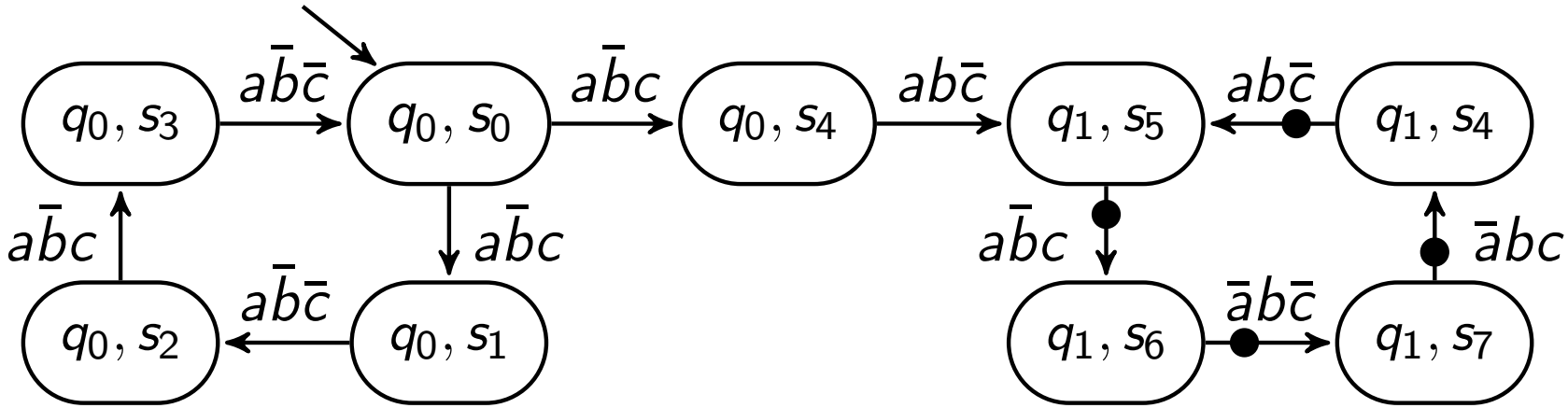


Explicit Approach



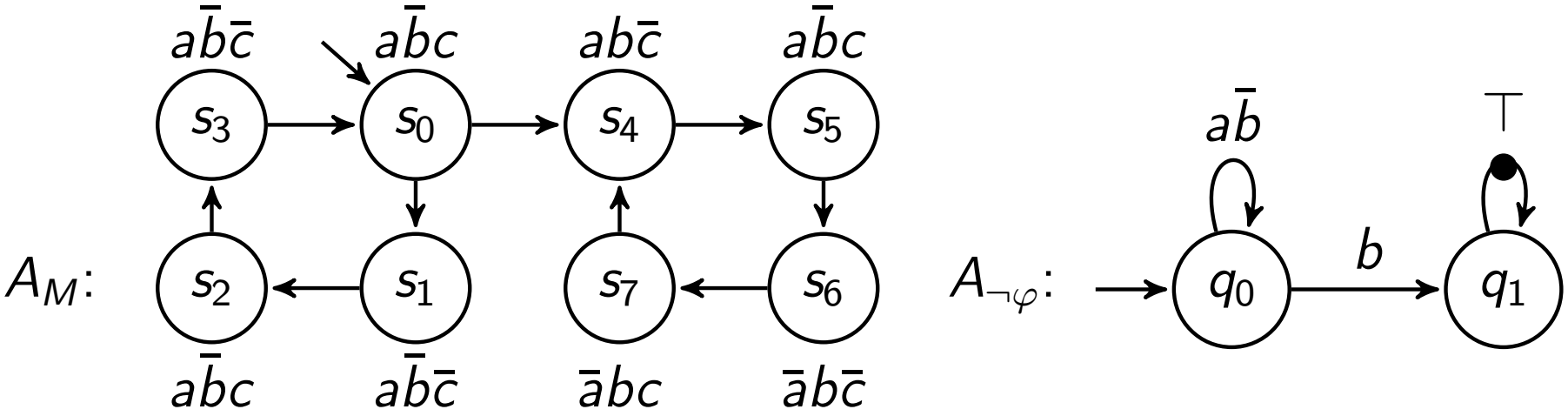


$A_{\neg\varphi} \otimes A_M$:



- Emptiness check = search for an accepting cycle in the product
- State explosion problem

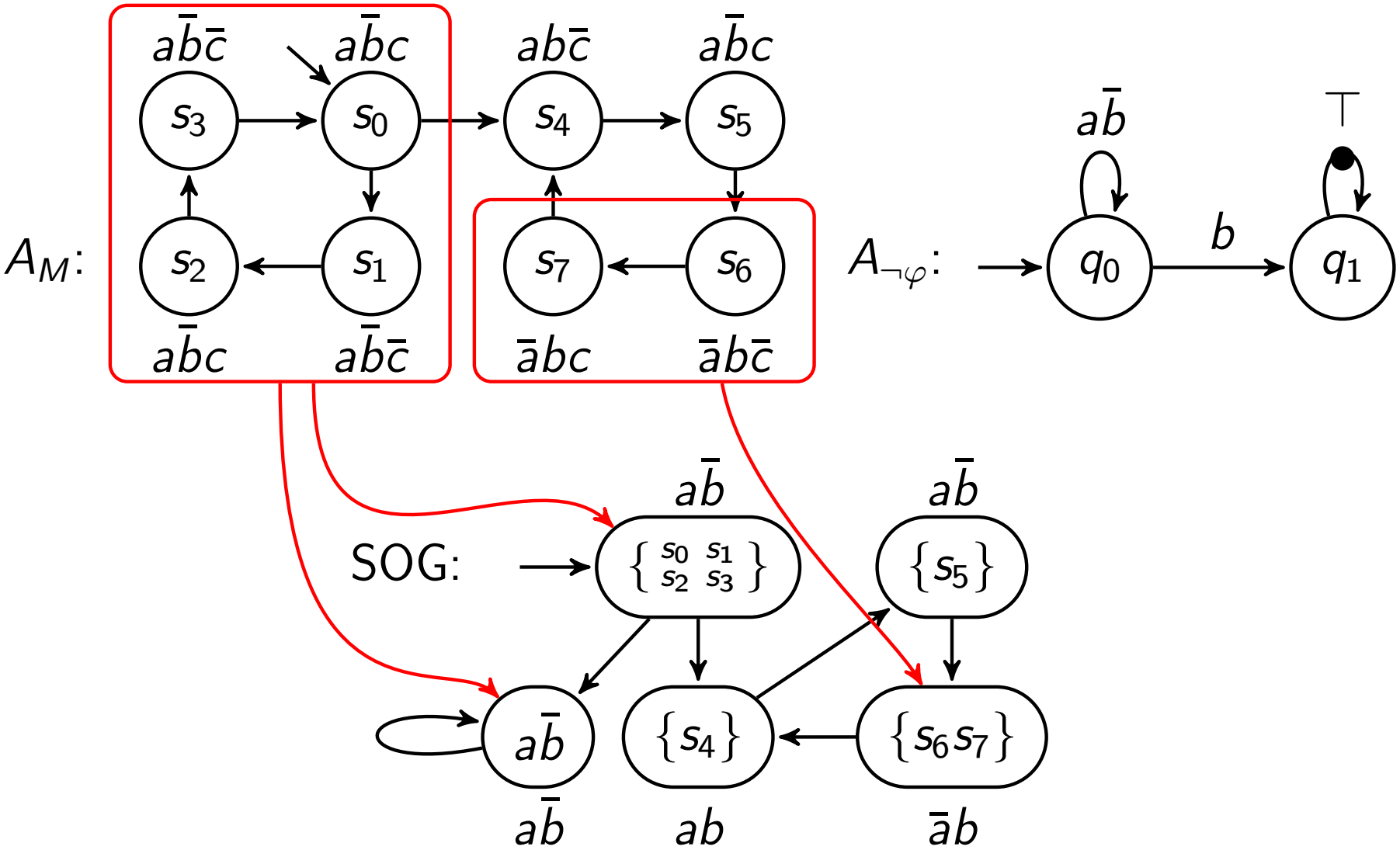
Symbolic Observation Graph (SOG)



- For stuttering invariant properties (e.g., LTL$\setminus \mathbf{X}$)
- Ignore non-observable propositions
- Aggregate Kripke states with homogeneous labels
- Represent aggregates using BDDs

K. Klai and D. Poitrenaud. MC-SOG: An LTL model checker based on symbolic observation graphs. In Proc. of PN'08, vol. 5062 of LNCS, pp. 288–306. Springer

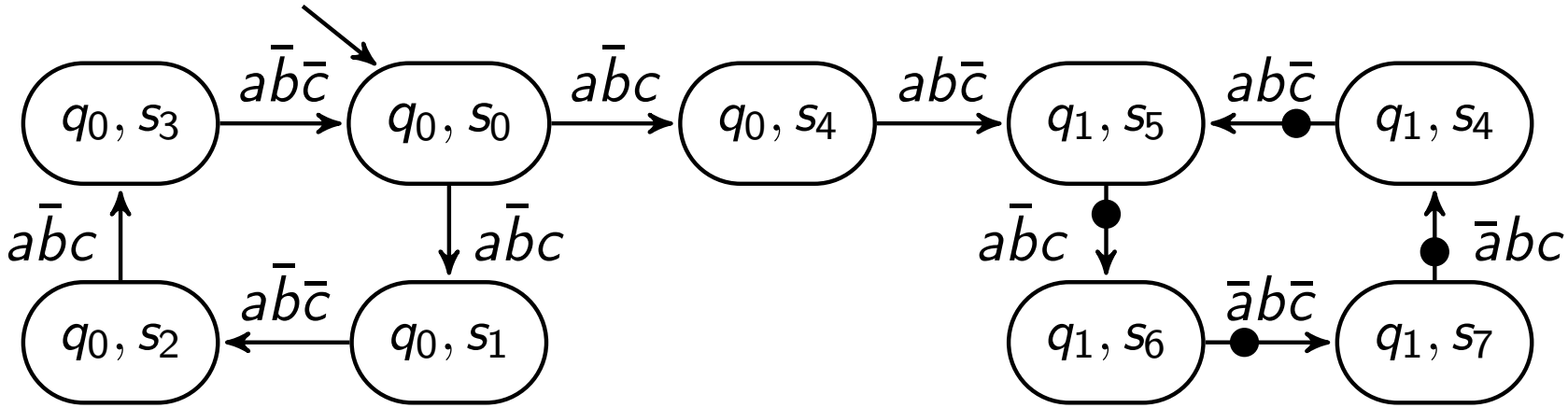
Symbolic Observation Graph (SOG)



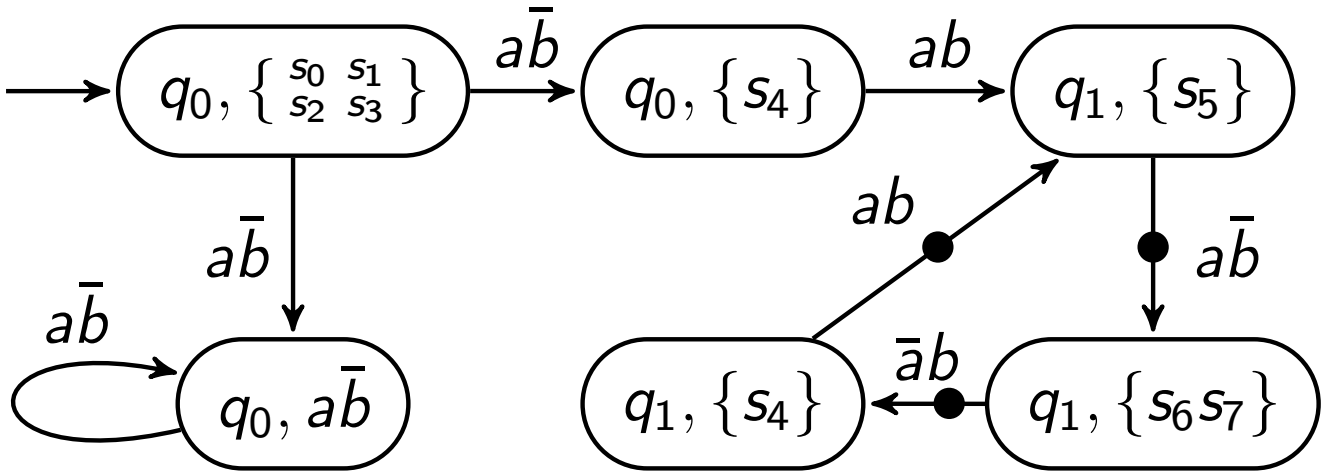
K. Klai and D. Poitrenaud. MC-SOG: An LTL model checker based on symbolic observation graphs. In Proc. of PN'08, vol. 5062 of LNCS, pp. 288–306. Springer

Product Sizes: Kripke vs. SOG

$A_{\neg\varphi} \otimes A_M$:



$A_{\neg\varphi} \otimes$ SOG:



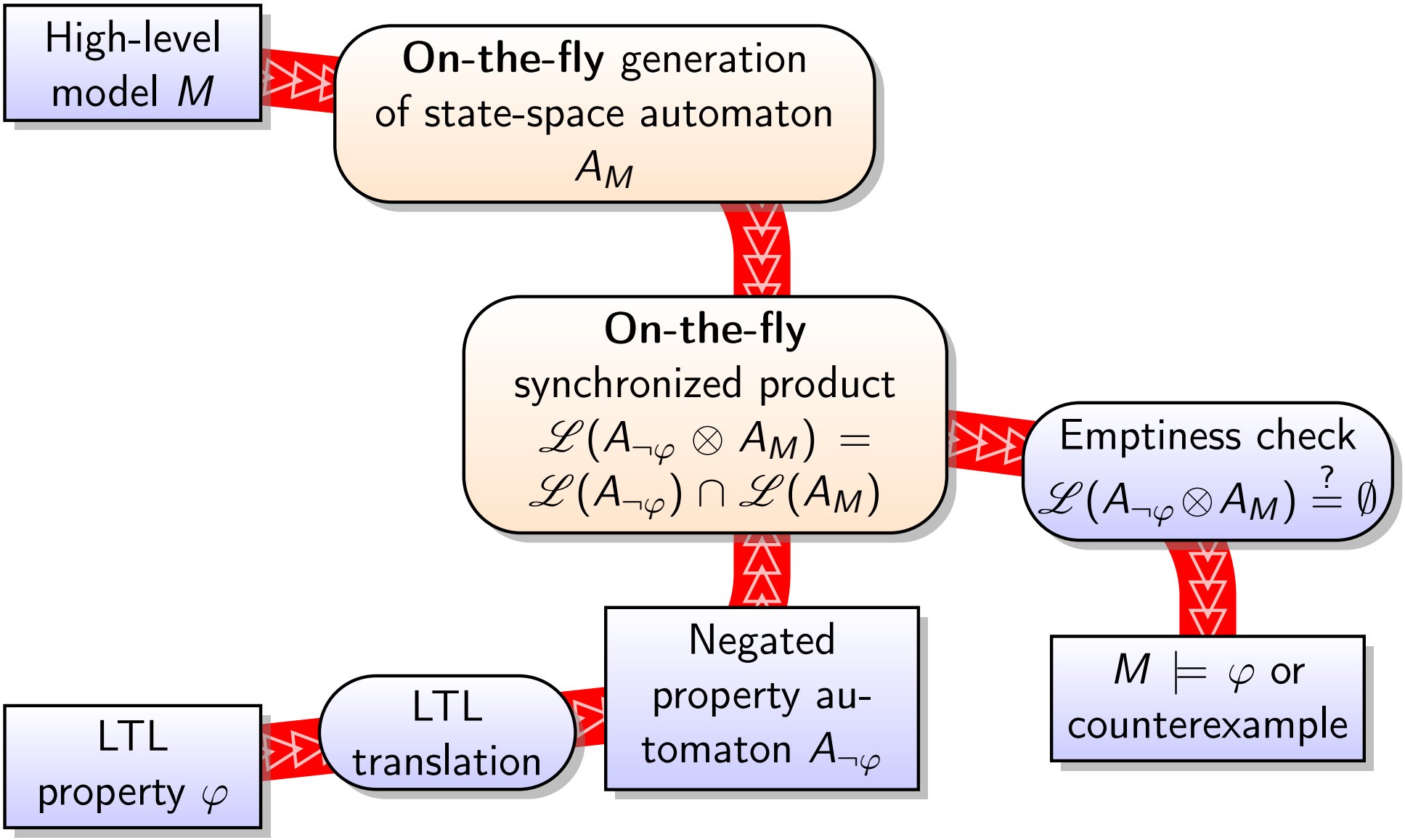
K. Klai and D. Poitrenaud. MC-SOG: An LTL model checker based on symbolic observation graphs. In Proc. of PN'08, vol. 5062 of LNCS, pp. 288–306. Springer

BCZ: Multiple-State Tableaux

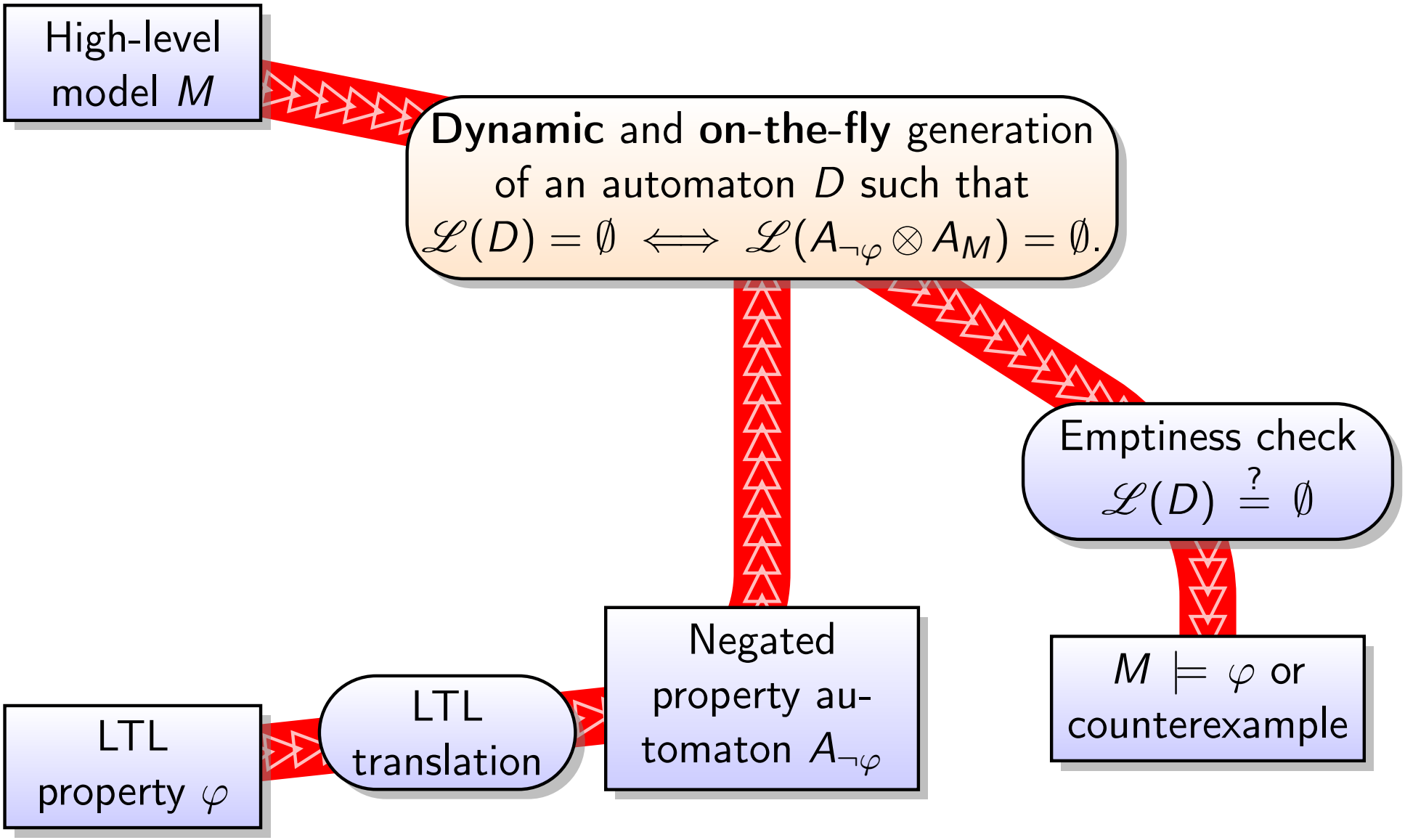
- Similar to SOG with only 1 step par aggregate:
  - supports full LTL
  - no need to search for livelock cycles
- Low aggregation power: on our example with low branching, it does not reduce the Kripke structure.

A. Biere, E. M. Clarke, and Y. Zhu. Multiple state and single state tableaux for combining local and global model checking. In Correct System Design, vol. 1710 of LNCS, pp. 163–179. Springer, 1999

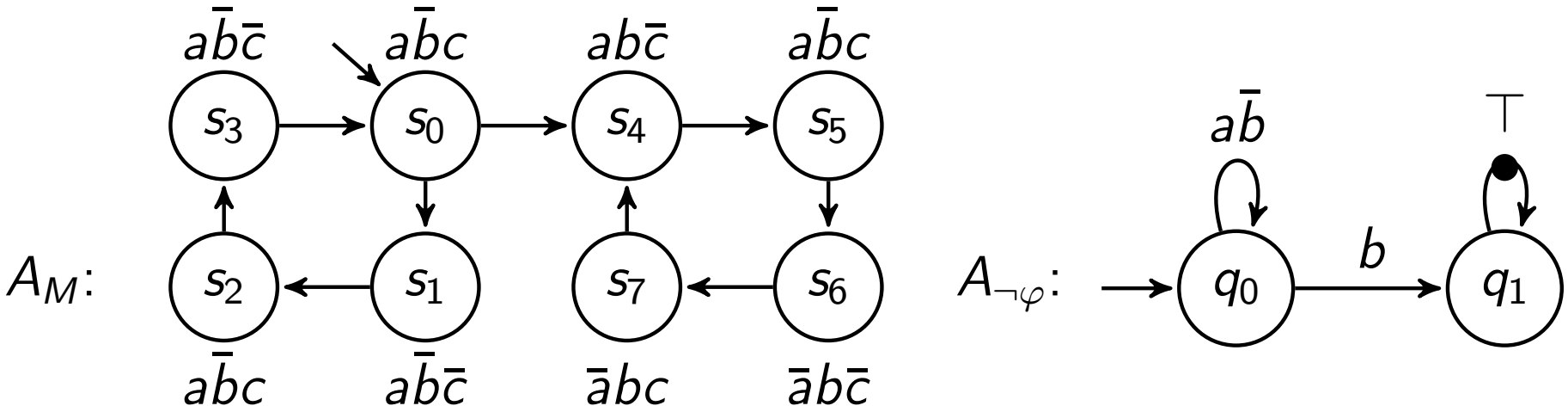
Building a Product Directly



Building a Product Directly

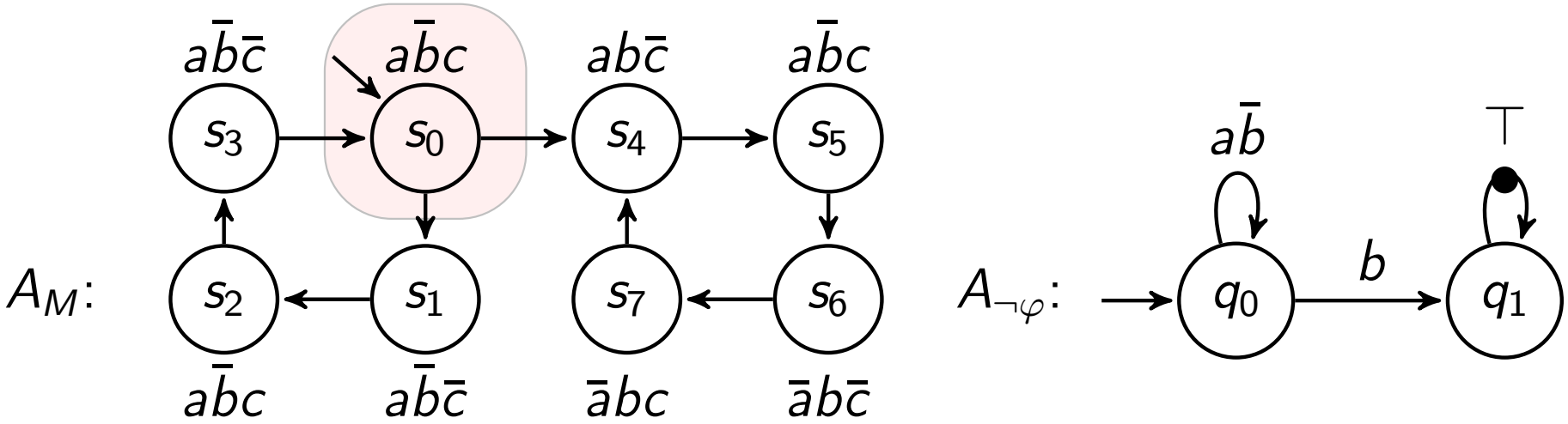


Self-Loop Aggregation Product (SLAP)



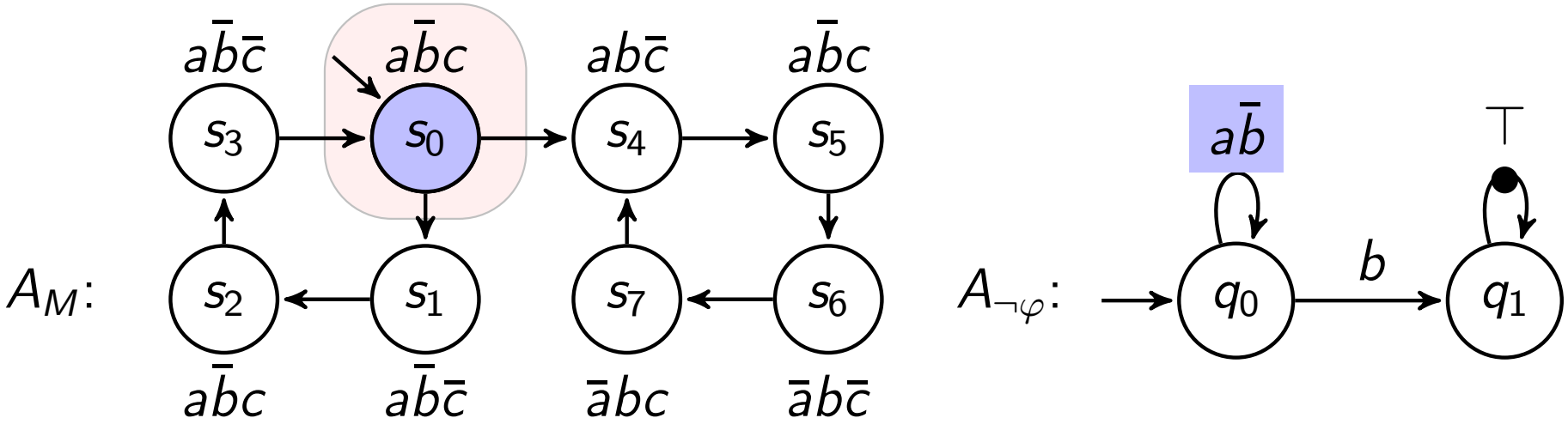
Self-Loop Aggregation **Product**:

Self-Loop Aggregation Product (SLAP)



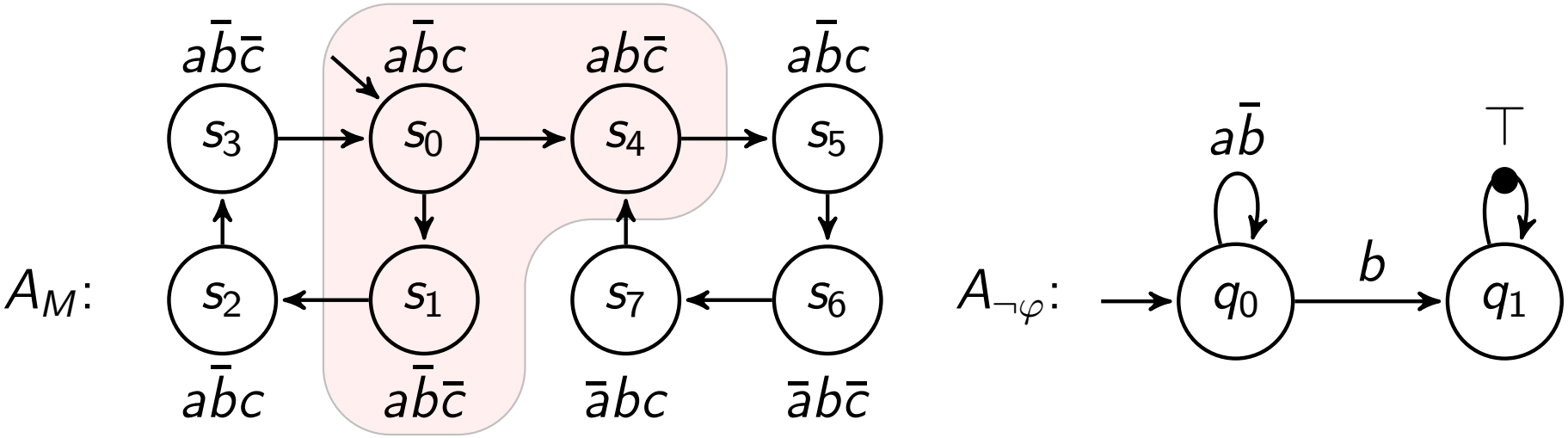
Self-Loop Aggregation **Product**:

Self-Loop Aggregation Product (SLAP)



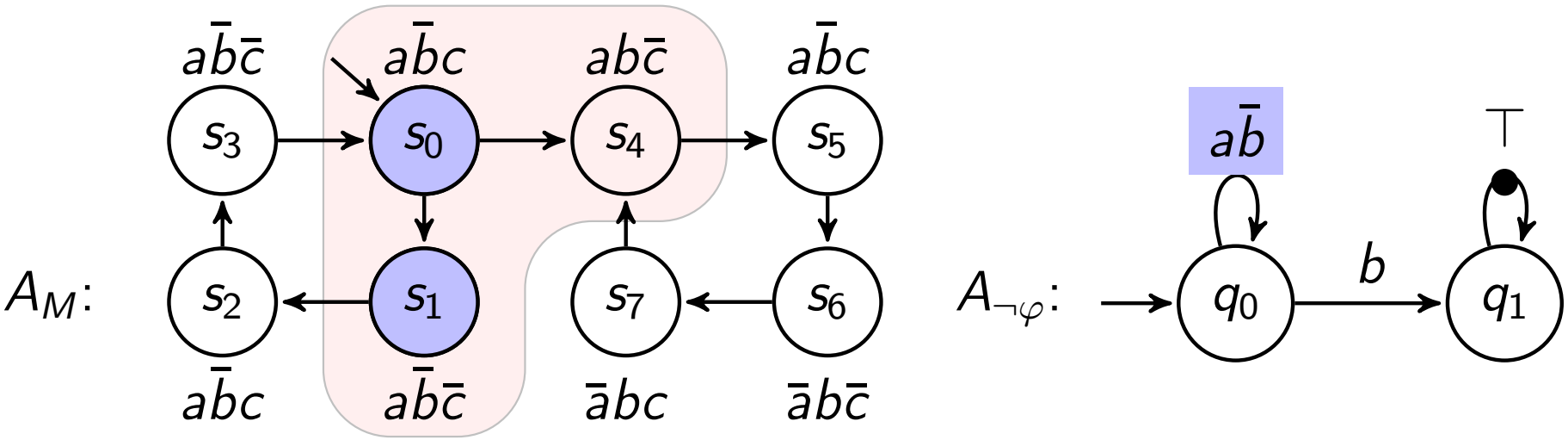
Self-Loop Aggregation **Product**:

Self-Loop Aggregation Product (SLAP)



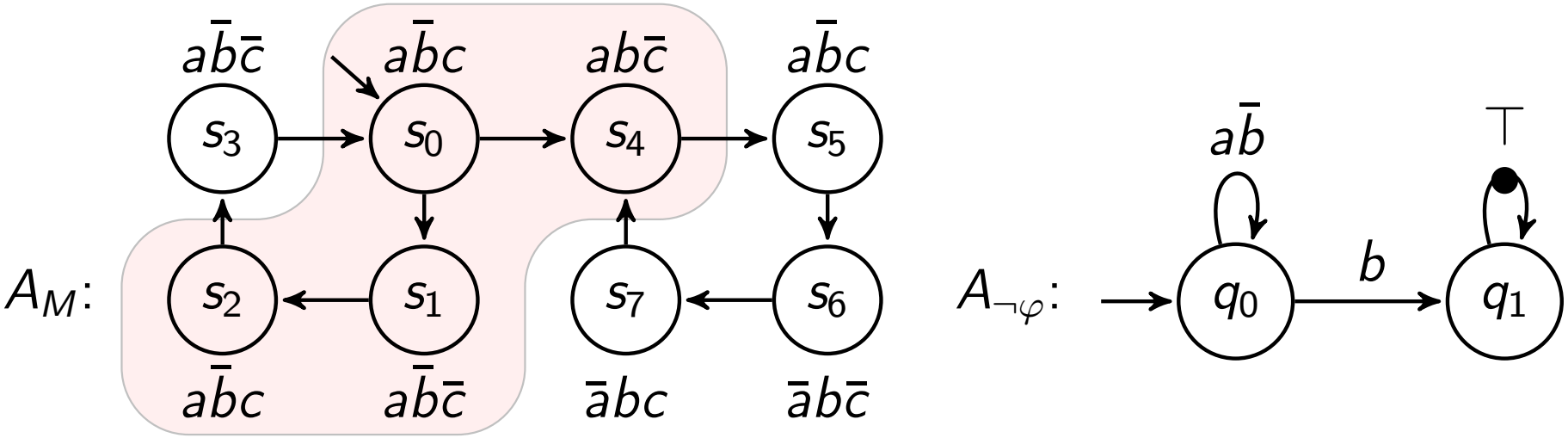
Self-Loop Aggregation **Product**:

Self-Loop Aggregation Product (SLAP)



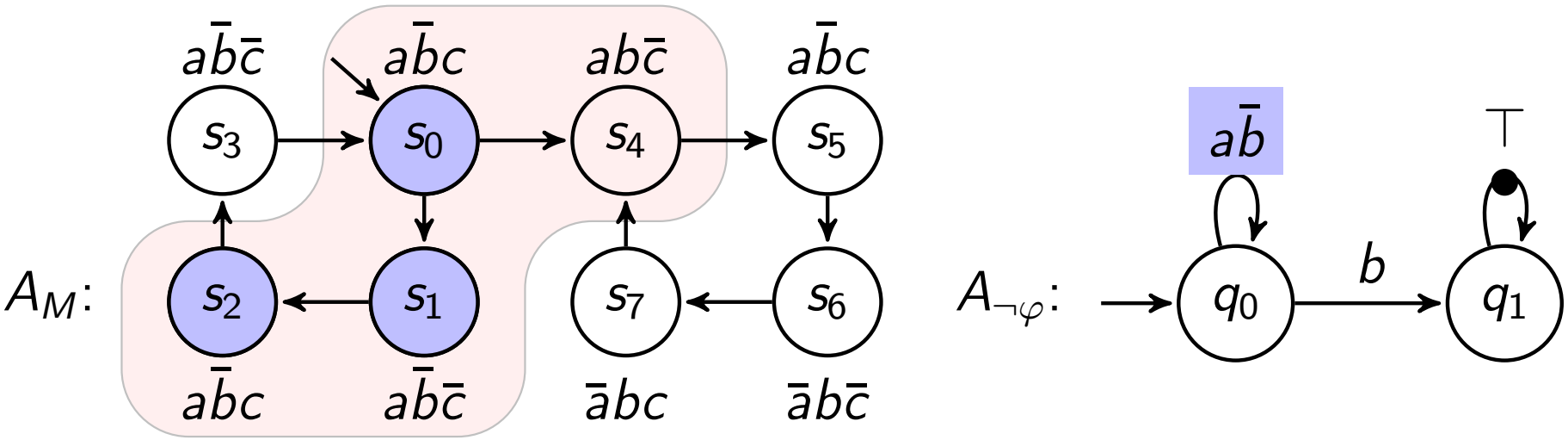
Self-Loop Aggregation **Product**:

Self-Loop Aggregation Product (SLAP)



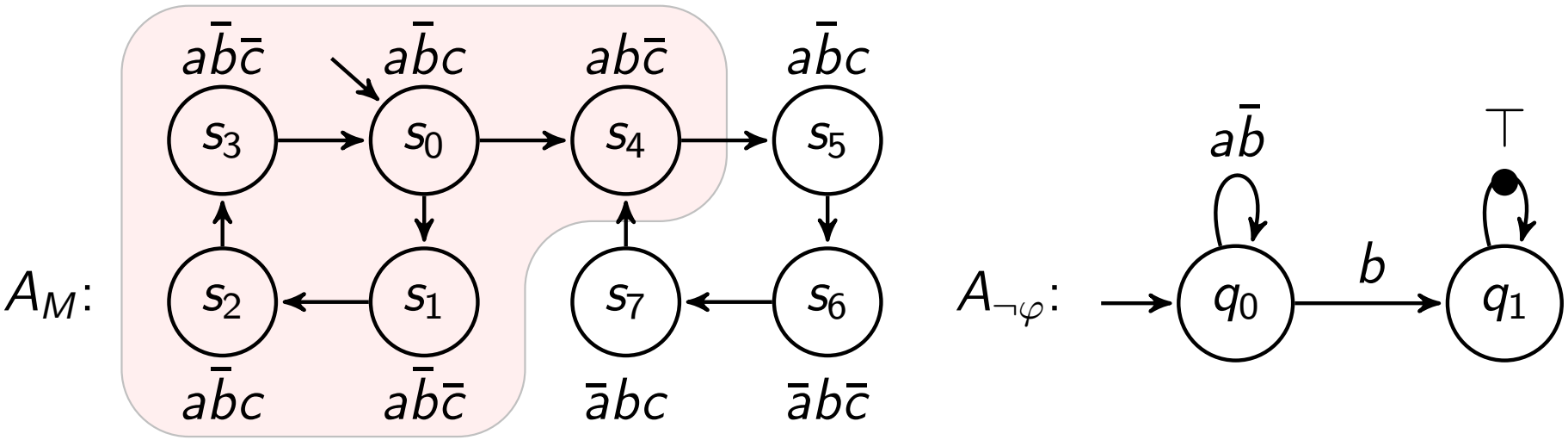
Self-Loop Aggregation **Product**:

Self-Loop Aggregation Product (SLAP)



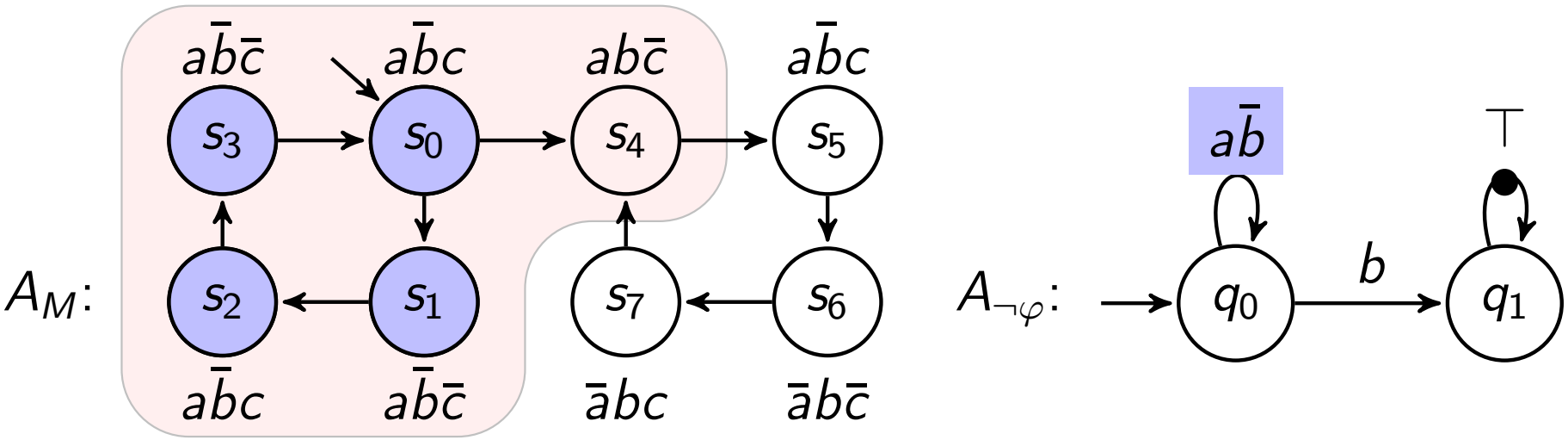
Self-Loop Aggregation **Product**:

Self-Loop Aggregation Product (SLAP)



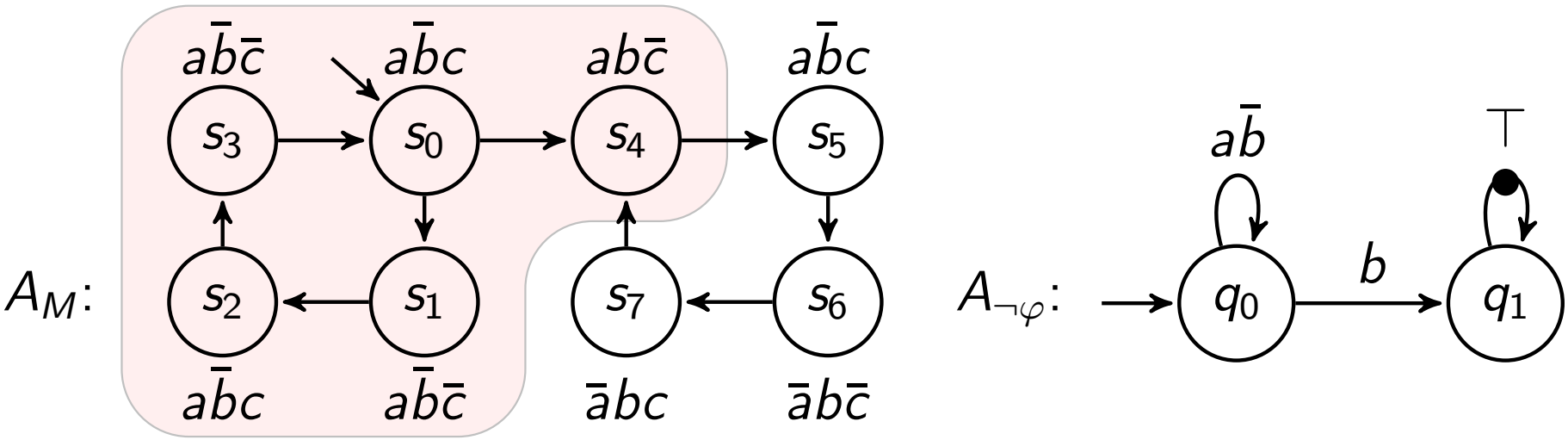
Self-Loop Aggregation **Product**:

Self-Loop Aggregation Product (SLAP)

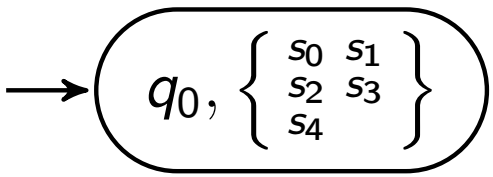


Self-Loop Aggregation **Product**:

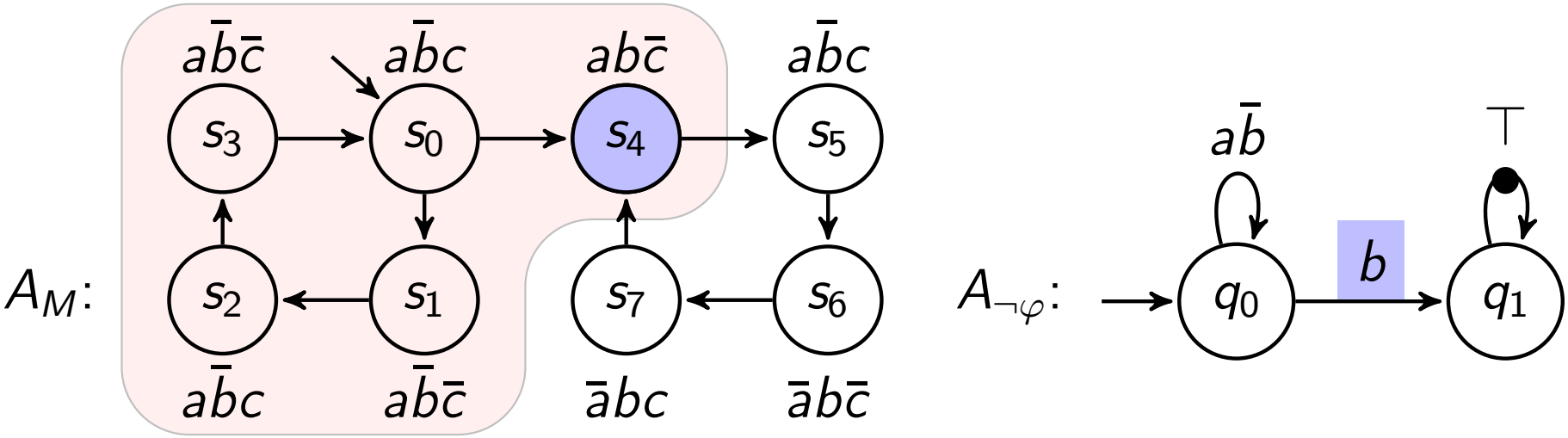
Self-Loop Aggregation Product (SLAP)



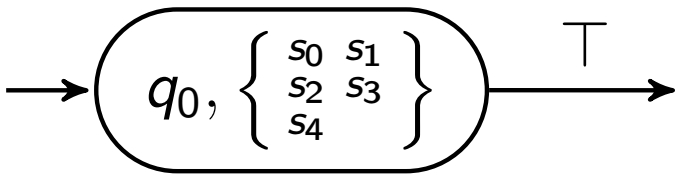
Self-Loop Aggregation **Product**:



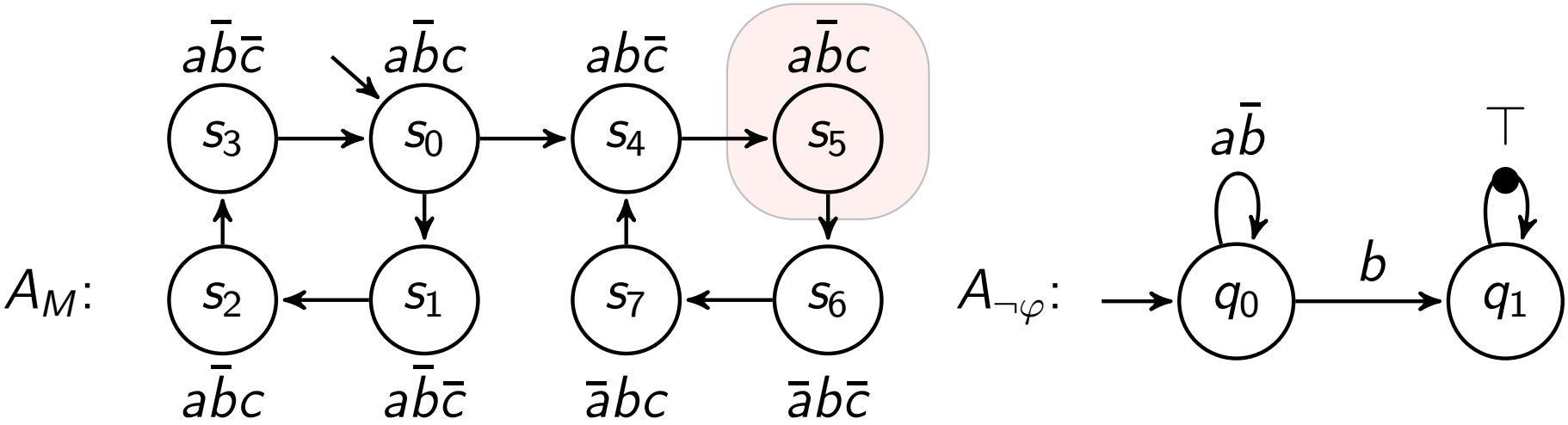
Self-Loop Aggregation Product (SLAP)



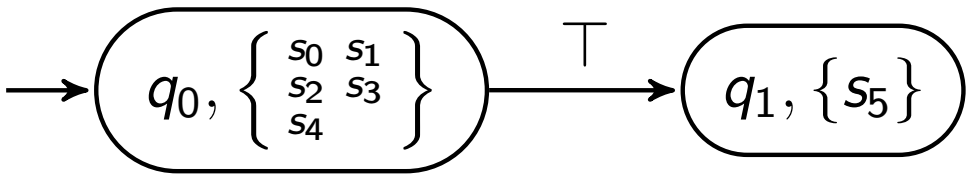
Self-Loop Aggregation **Product**:



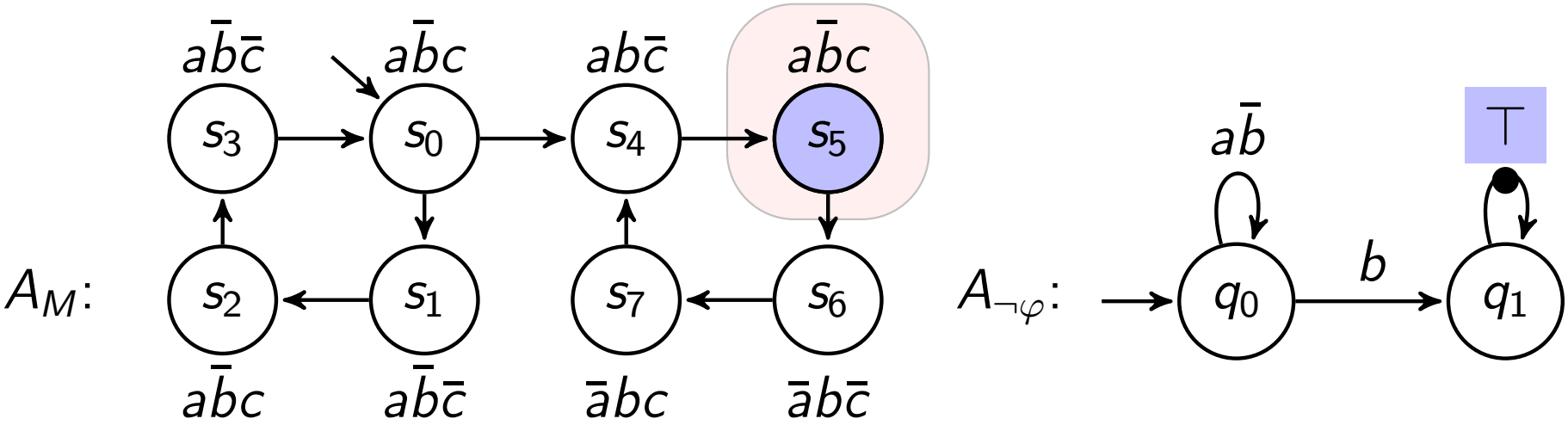
Self-Loop Aggregation Product (SLAP)



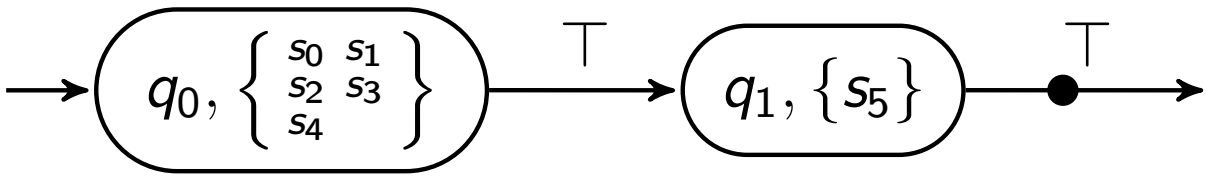
Self-Loop Aggregation **Product**:



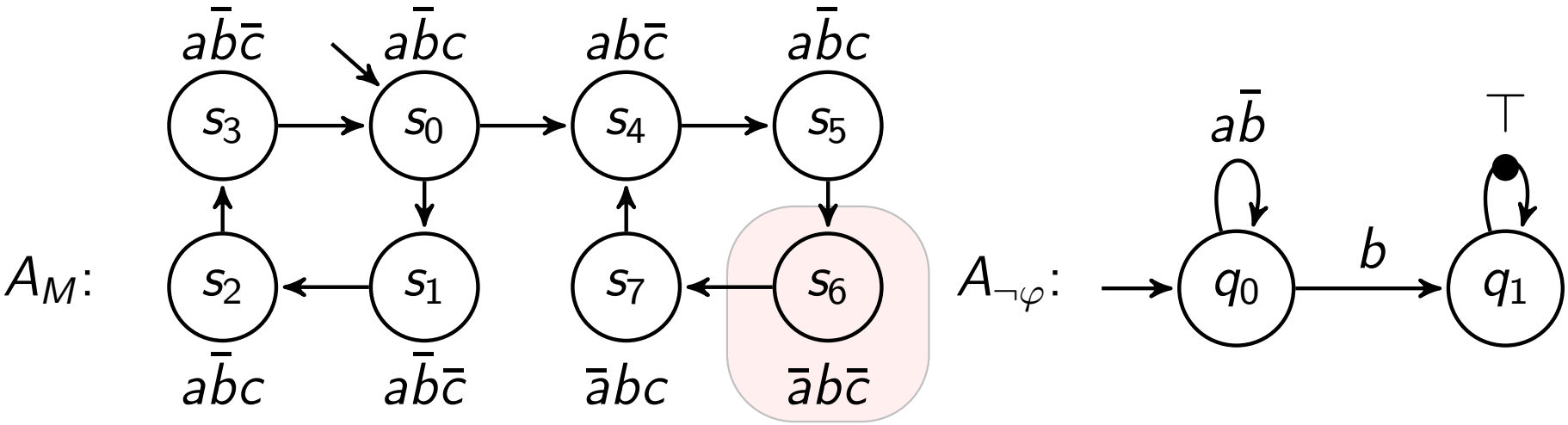
Self-Loop Aggregation Product (SLAP)



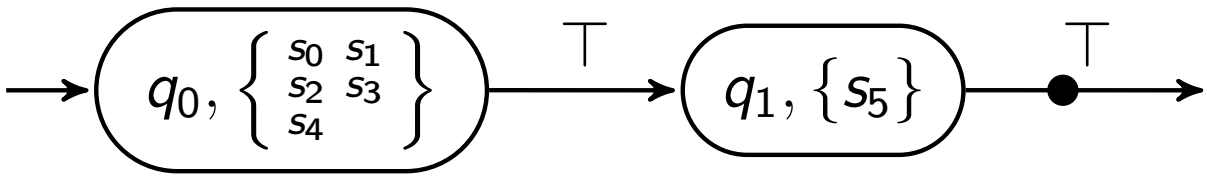
Self-Loop Aggregation **Product**:



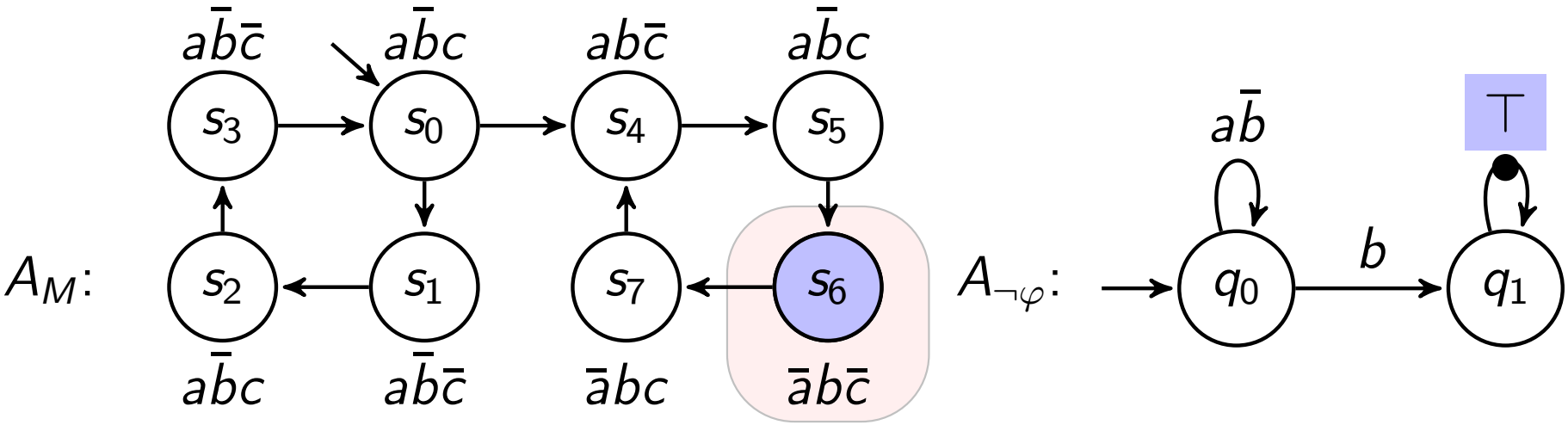
Self-Loop Aggregation Product (SLAP)



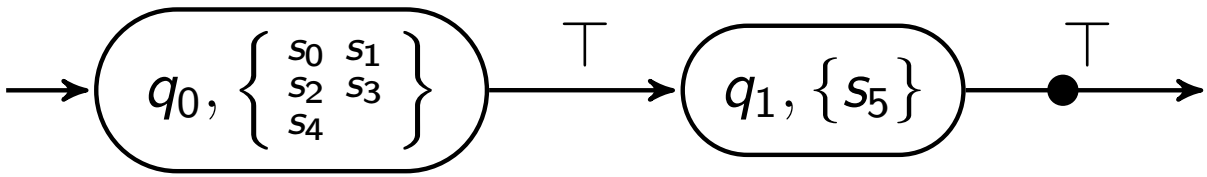
Self-Loop Aggregation **Product**:



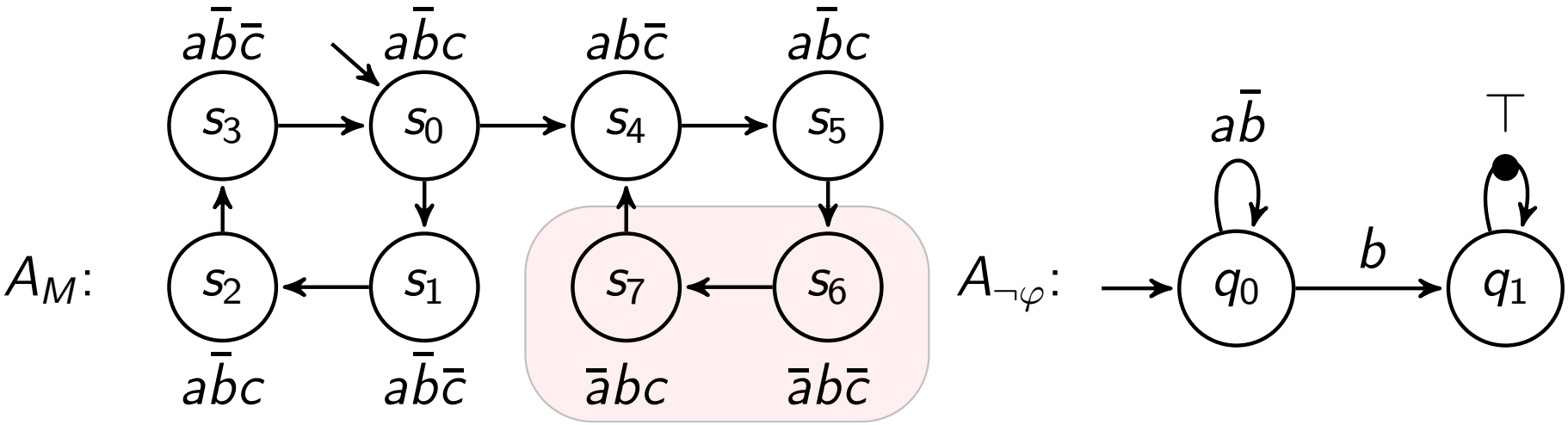
Self-Loop Aggregation Product (SLAP)



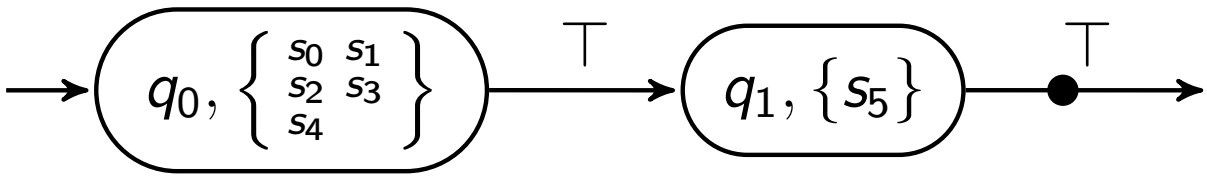
Self-Loop Aggregation **Product**:



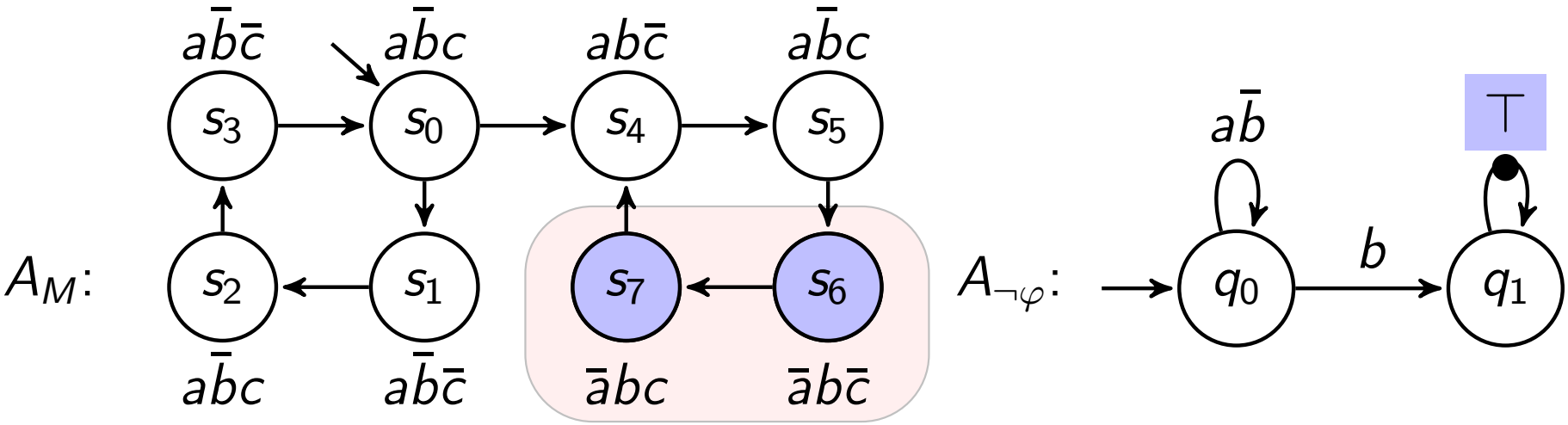
Self-Loop Aggregation Product (SLAP)



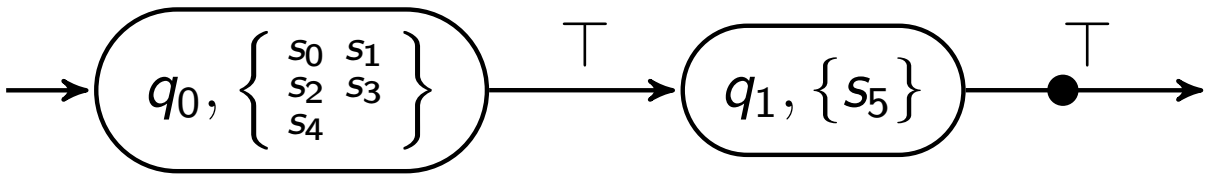
Self-Loop Aggregation **Product**:



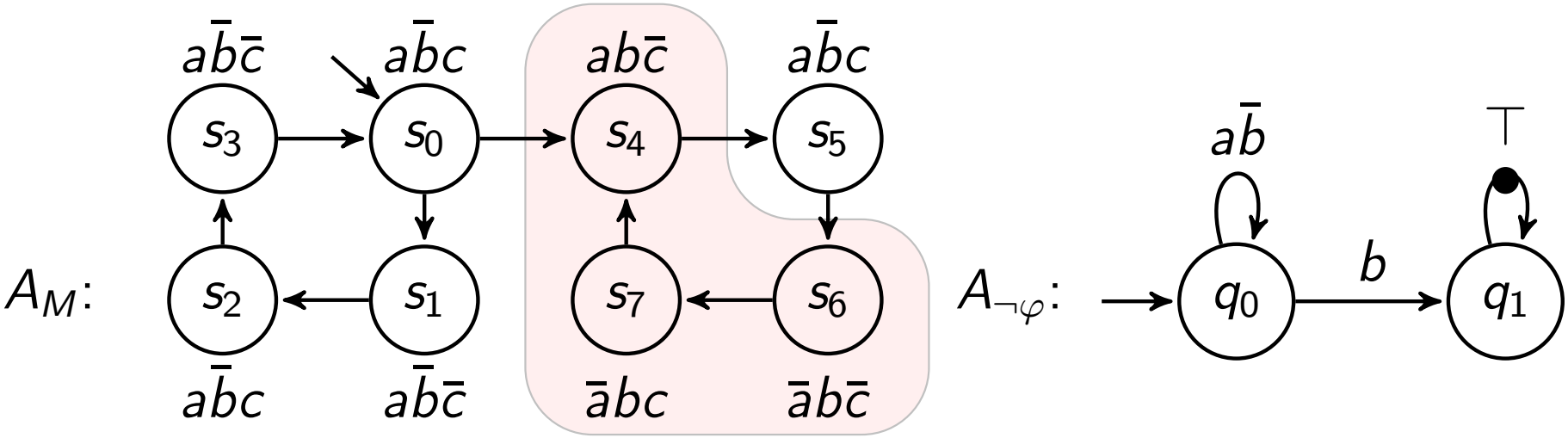
Self-Loop Aggregation Product (SLAP)



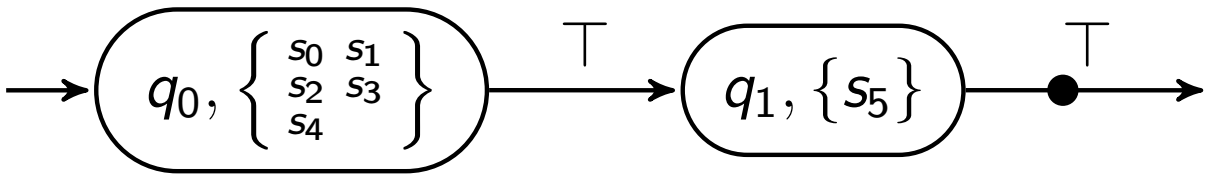
Self-Loop Aggregation **Product**:



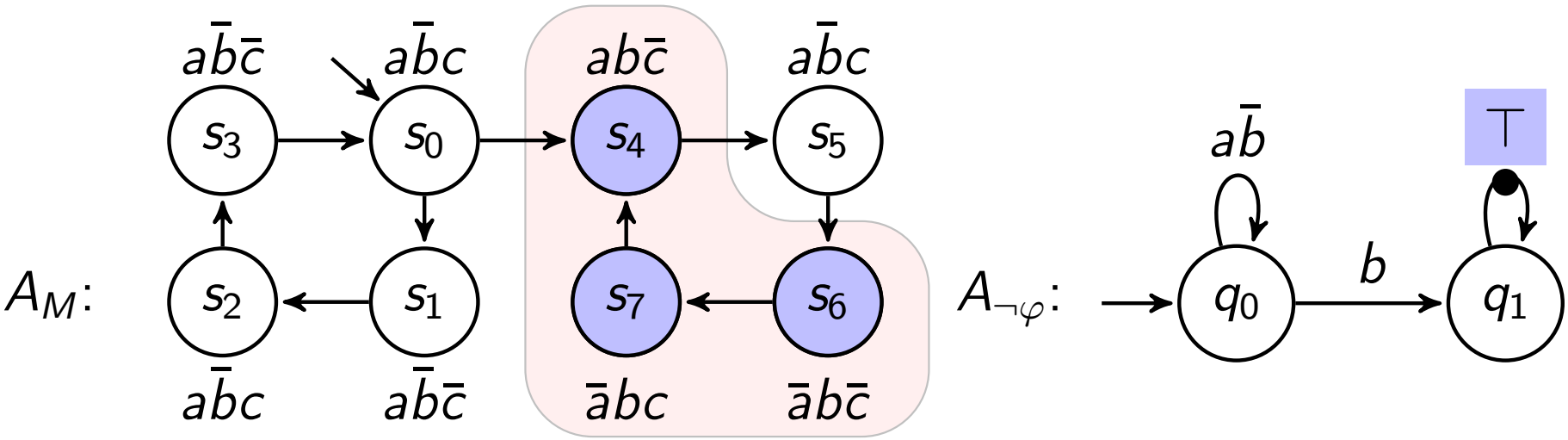
Self-Loop Aggregation Product (SLAP)



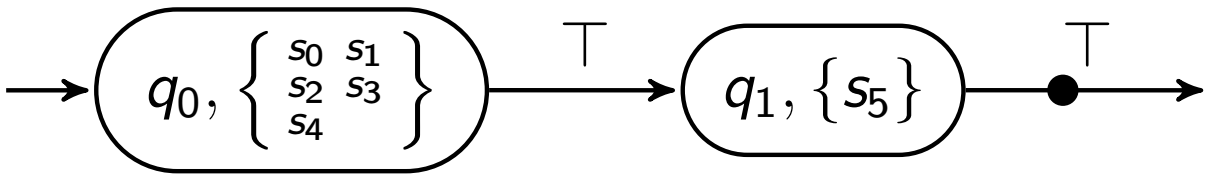
Self-Loop Aggregation **Product**:



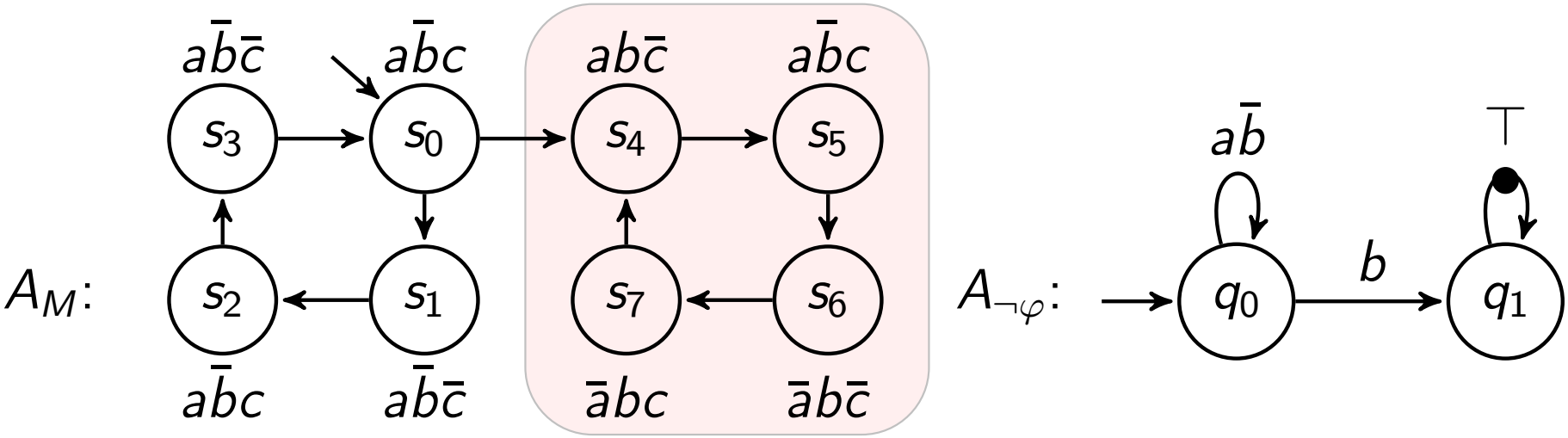
Self-Loop Aggregation Product (SLAP)



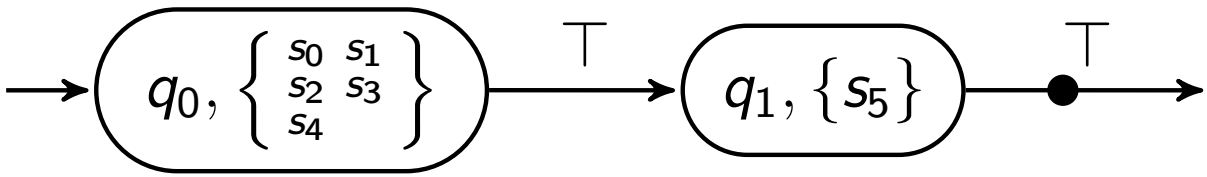
Self-Loop Aggregation **Product**:



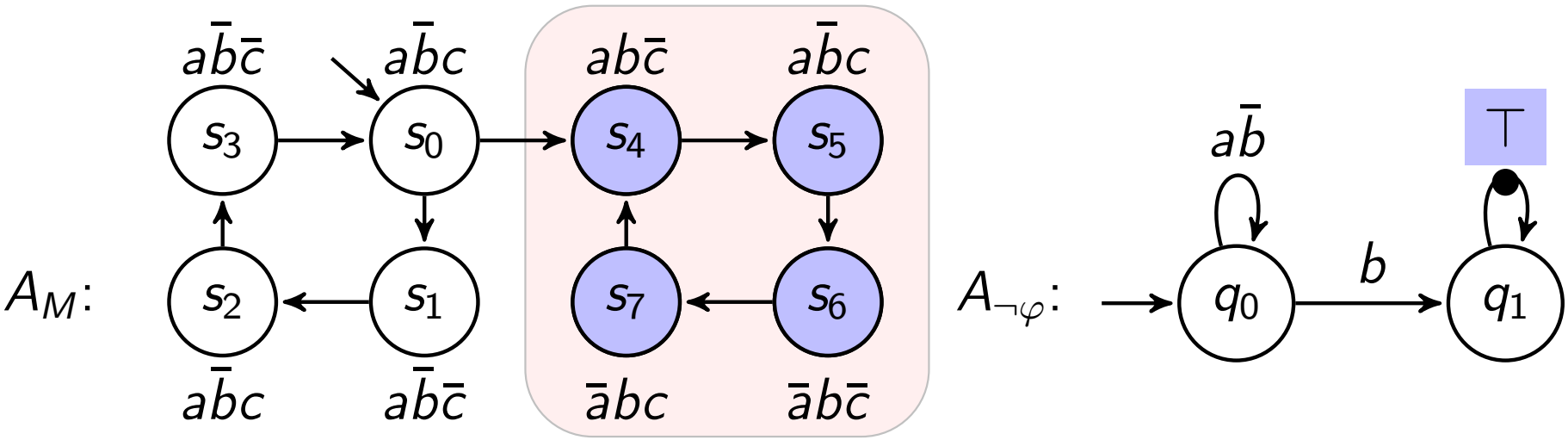
Self-Loop Aggregation Product (SLAP)



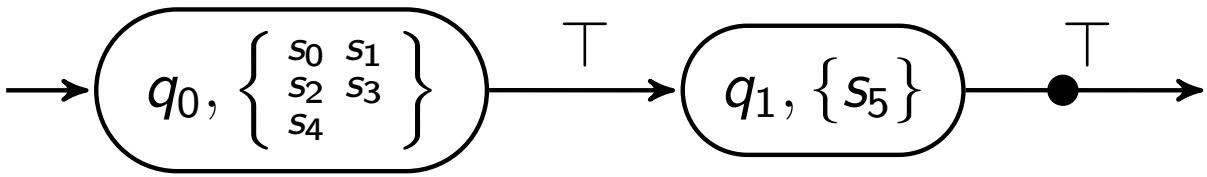
Self-Loop Aggregation **Product**:



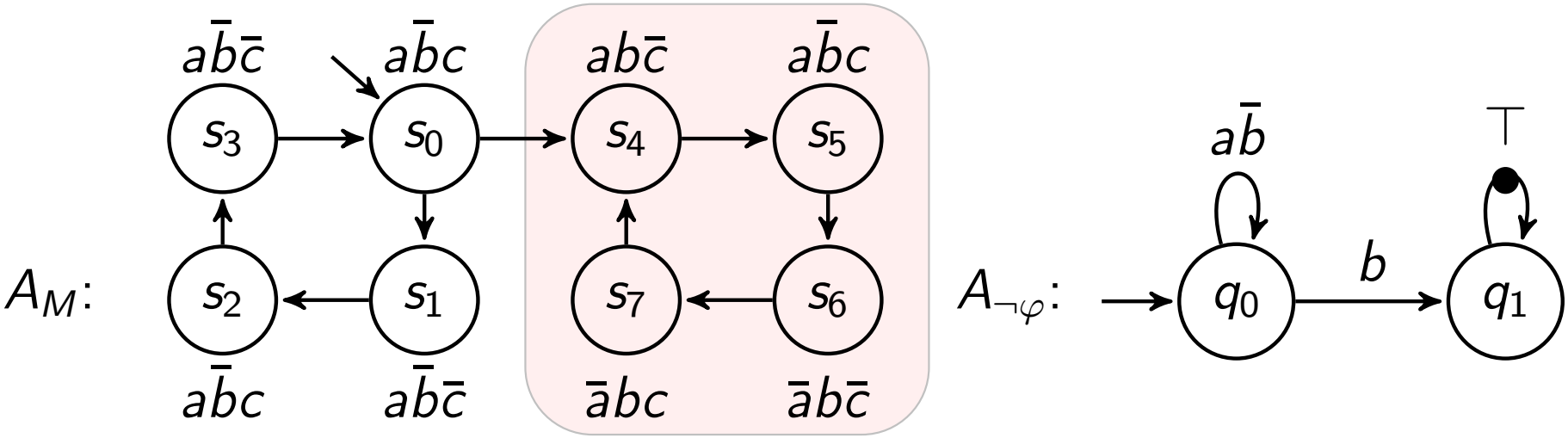
Self-Loop Aggregation Product (SLAP)



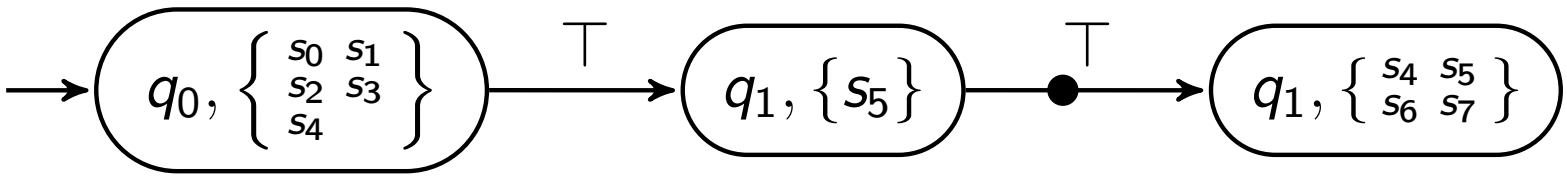
Self-Loop Aggregation **Product**:



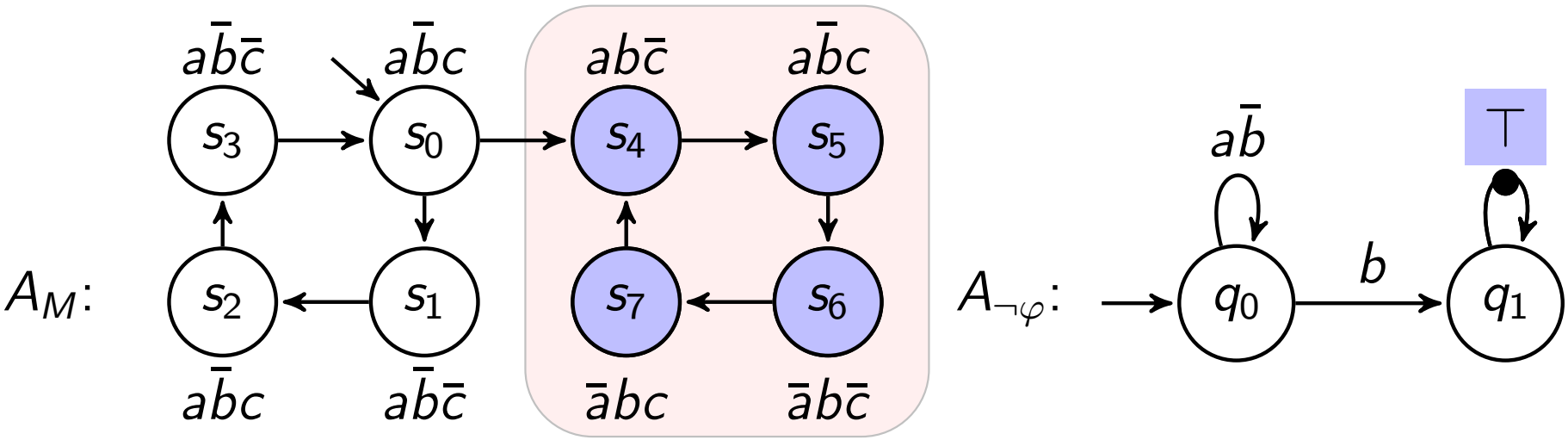
Self-Loop Aggregation Product (SLAP)



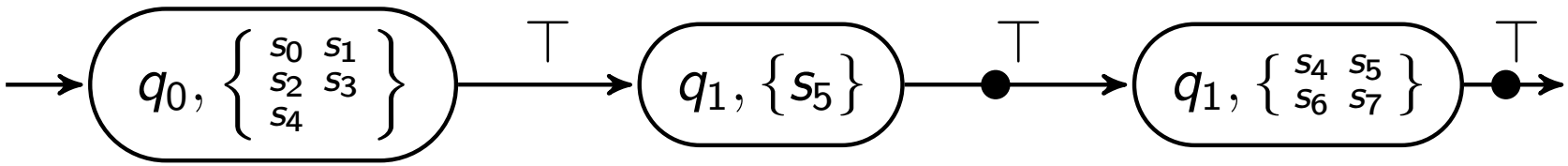
Self-Loop Aggregation **Product**:



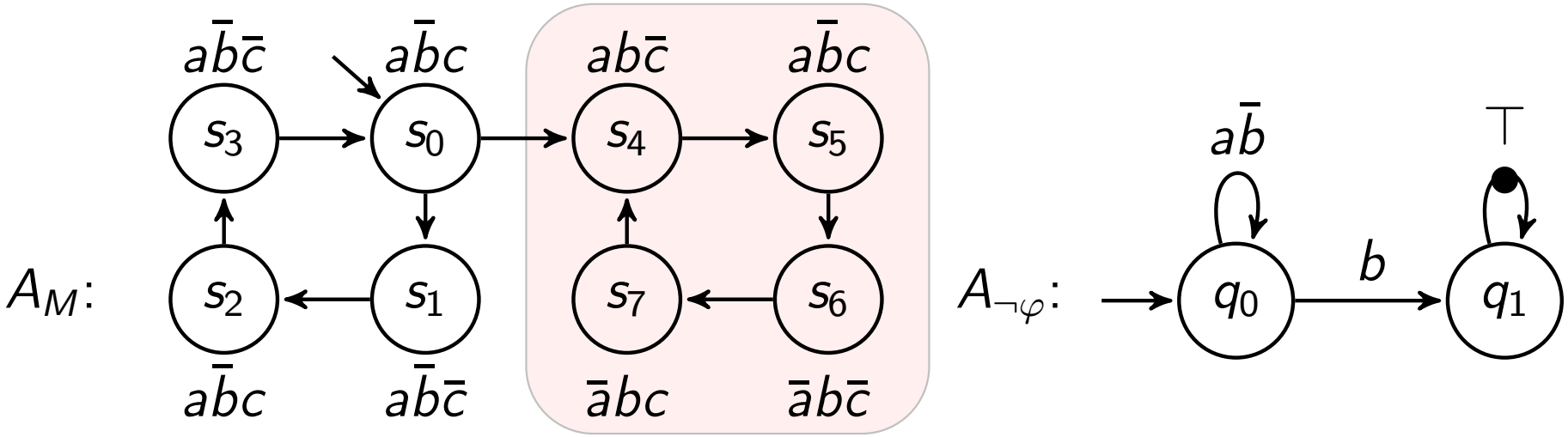
Self-Loop Aggregation Product (SLAP)



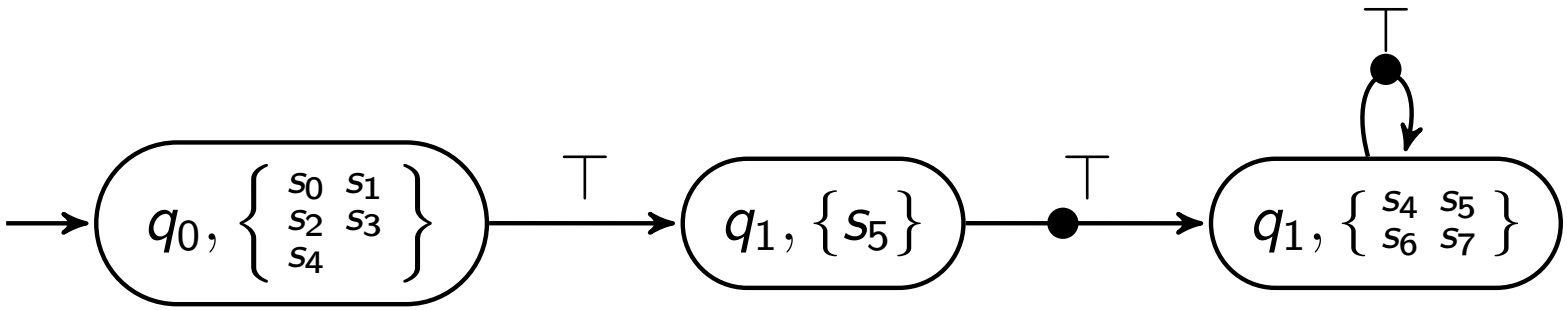
Self-Loop Aggregation **Product**:



Self-Loop Aggregation Product (SLAP)



Self-Loop Aggregation **Product**:



Fully Symbolic Approach (FS)

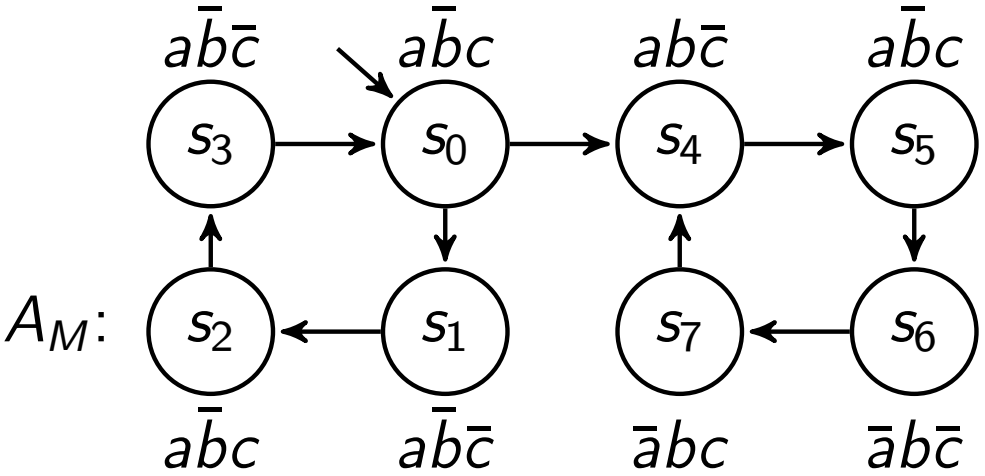
- Encode the Kripke structure and the property automaton using BDDs.
- Combine the two BDDs. (Symbolic product.)
- Use fixpoint computations to decide whether the product contains an accepting cycle.
- Two symbolic emptiness checks compared:
  - EL
  - OWCTY

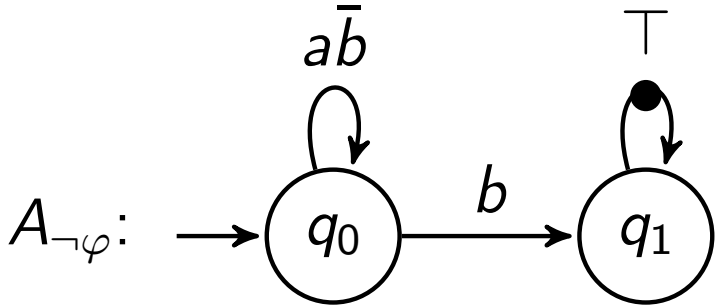

K. Fisler, R. Fraer, G. Kamhi, M. Y. Vardi, and Z. Yang. Is there a best symbolic cycle-detection algorithm? In Proc. of TACAS'01, vol. 2031 of LNCS, pp. 420–434. Springer

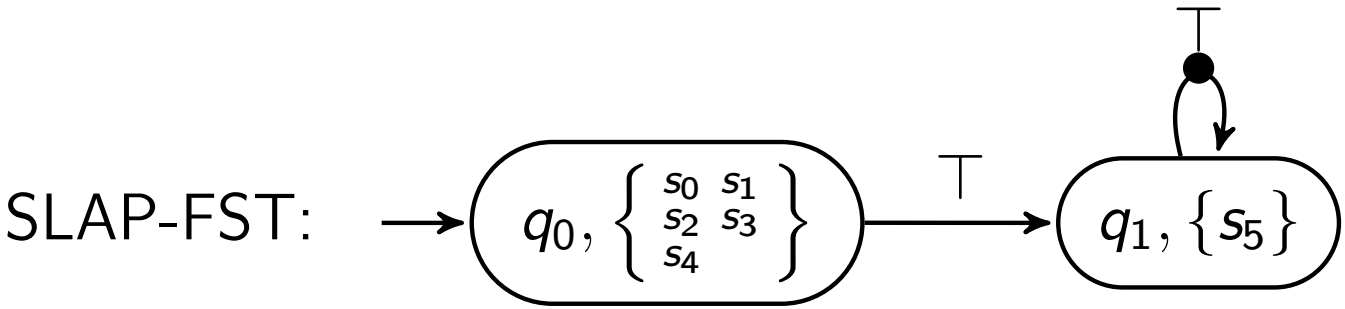

F. Somenzi, K. Ravi, and R. Bloem. Analysis of symbolic SCC hull algorithms. In Proc. of FMCAD'02, vol. 2517 of LNCS, pp. 88–105. Springer

SLAP-FST

- FST = Fully Symbolic search in Terminal automata states
- When reaching a state of $A_{\neg\varphi}$ which is terminal (and accepting) use a fully symbolic search.







Summary of methods

	Explicit	BCZ	SOG	SLAP	PDP	FS
logic	LTL		LTL$\setminus$ **X**	LTL		
prop. aut.	explicit					symb./expl.
data str.	graph	BDD+graph			BDD[]	BDD
empt. chk	explicit				symbolic (OWCTY/EL)	

PDP = Property Driven Partitioning,

R. Sebastiani, S. Tonetta, and M. Y. Vardi. Symbolic systems, explicit properties: on hybrid approches for LTL symbolic model checking. In Proc. of CAV'05, vol. 3576 of LNCS, pp. 350–363. Springer

Experimental Framework

Implemented algorithms

BCZ, SOG, SLAP, SLAP-FST, EL, OWCTY

Tools used

SDD Hierarchical Set Decision Diagrams (`http://ddd.lip6.fr/`)

- Hierarchy (memory gain)
- Automatic saturation (improves fixpoint computations)

Spot Model checking library (`http://spot.lip6.fr/`)

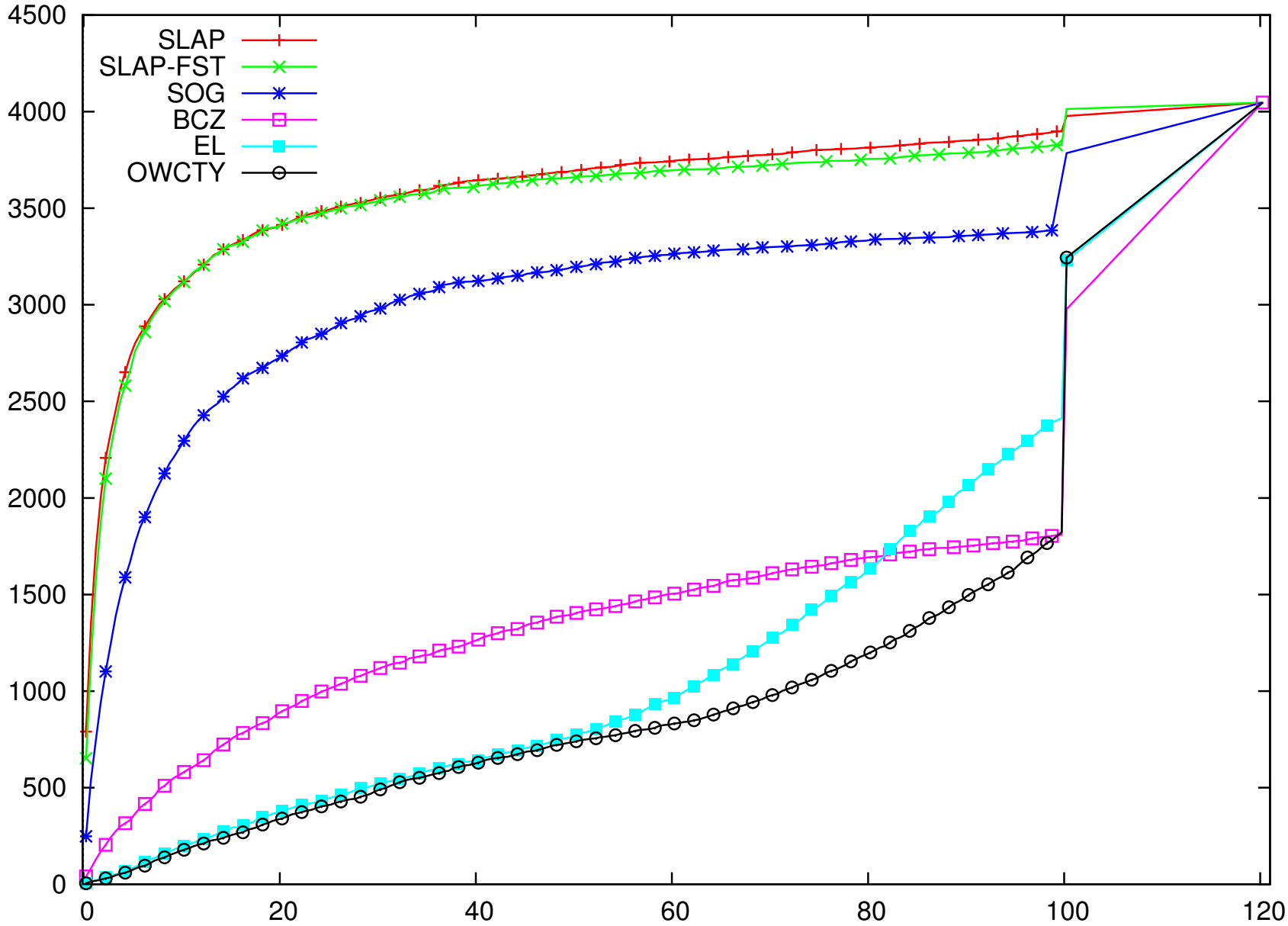
- Good LTL-to-TGBA translation
- Explicit emptiness-check algorithms

Data

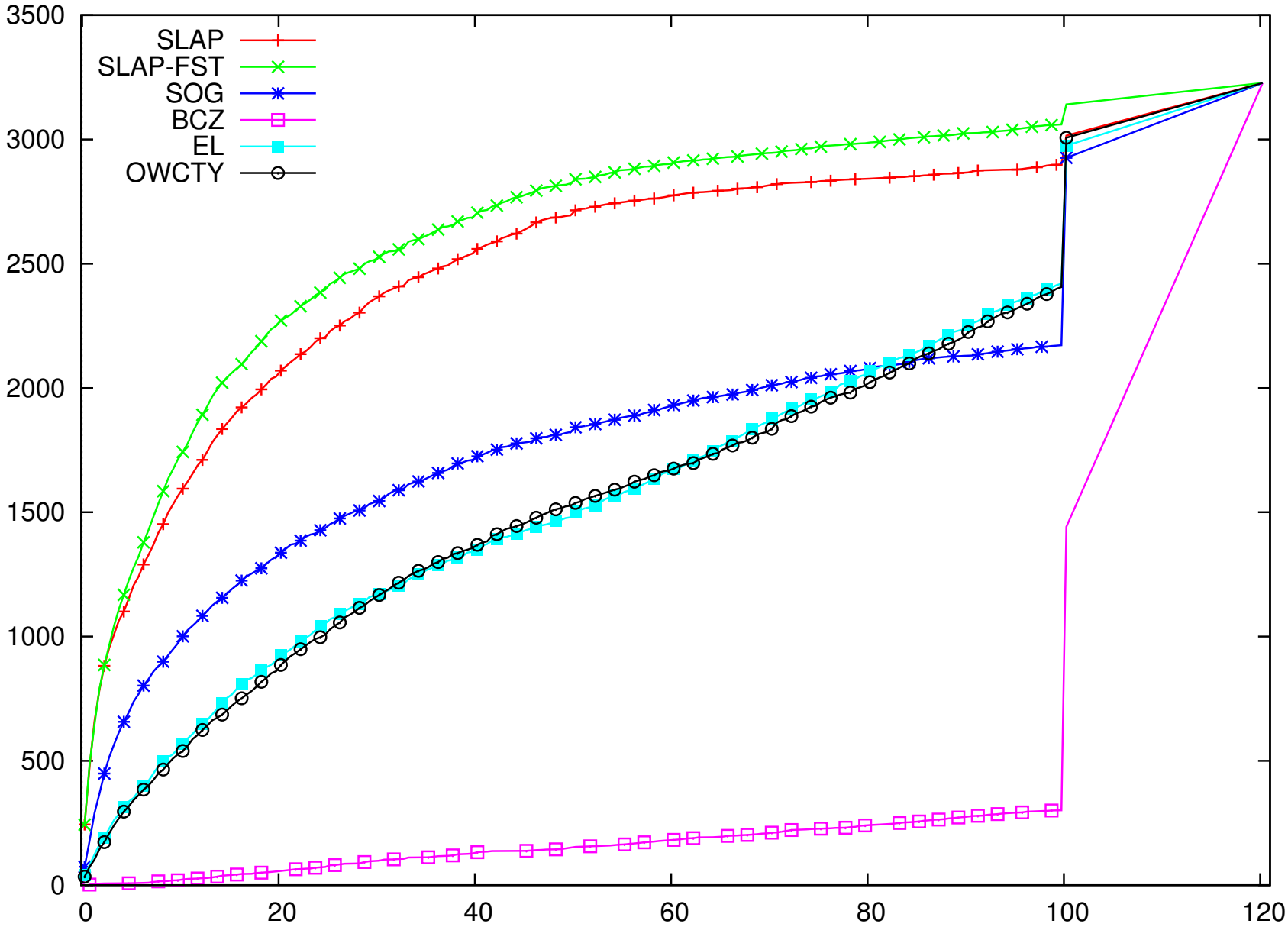
Models Scalable toy models with $10^6 \ldots 10^{66}$ states.

Formulas Random formulas (filtered), plus a set of (random) weak fairness properties.

Cumulative Plot: Violated Formulas

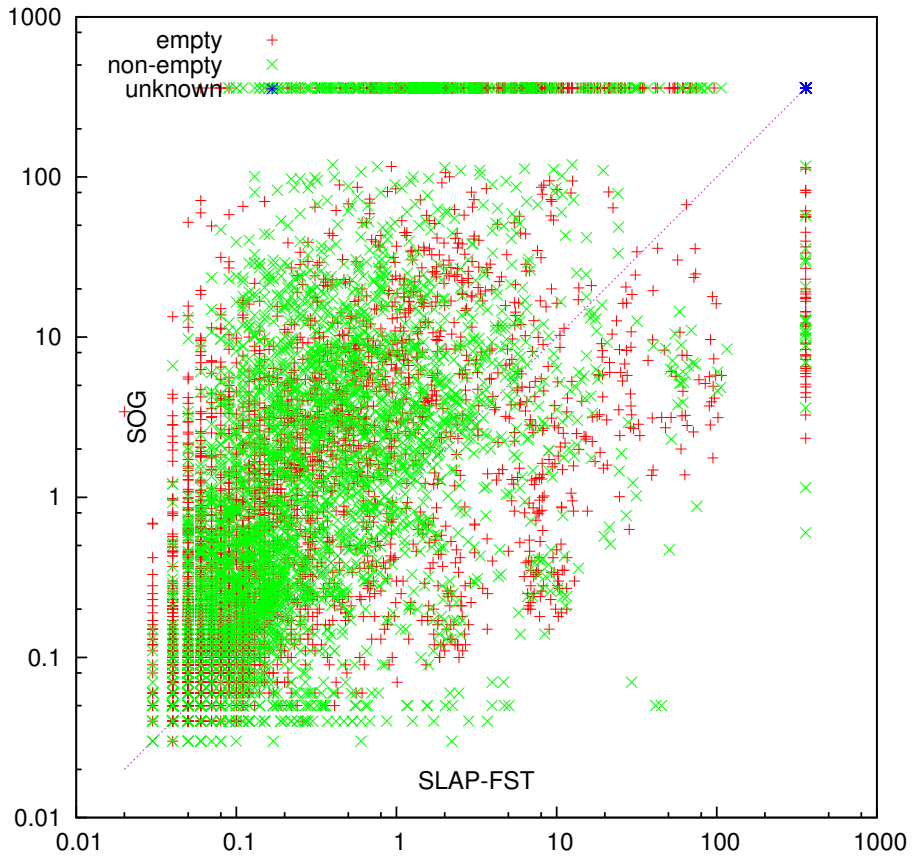


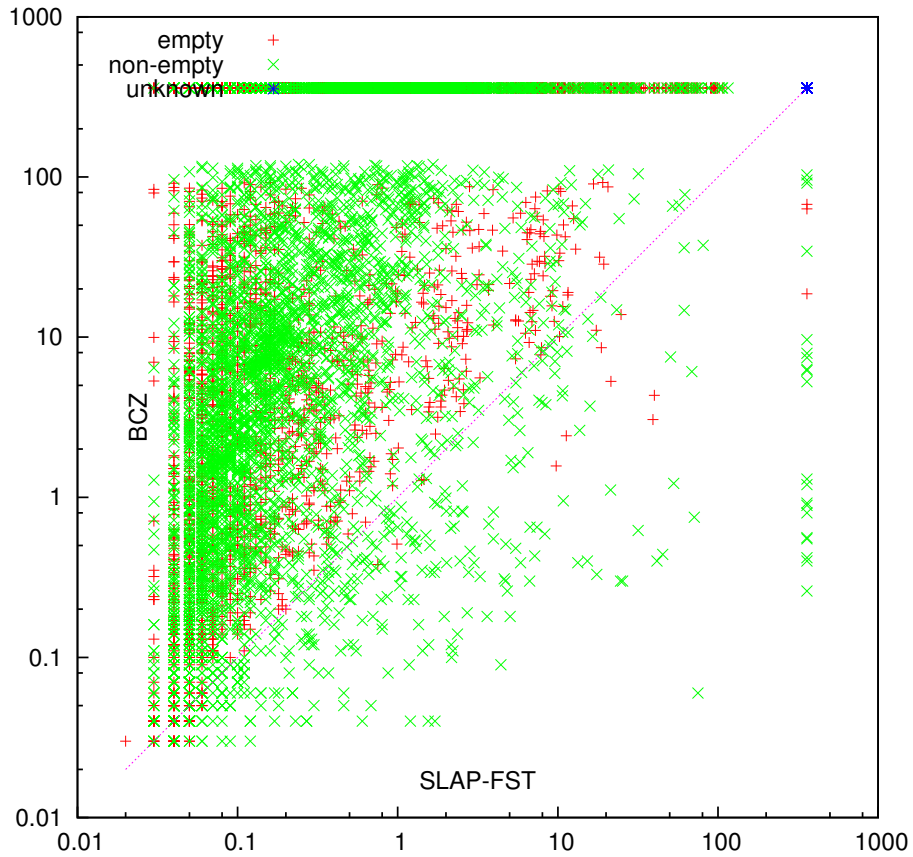
Cumulative Plot: Verified Formulas



Scatter Plots: SLAP-FST vs. Hybrid

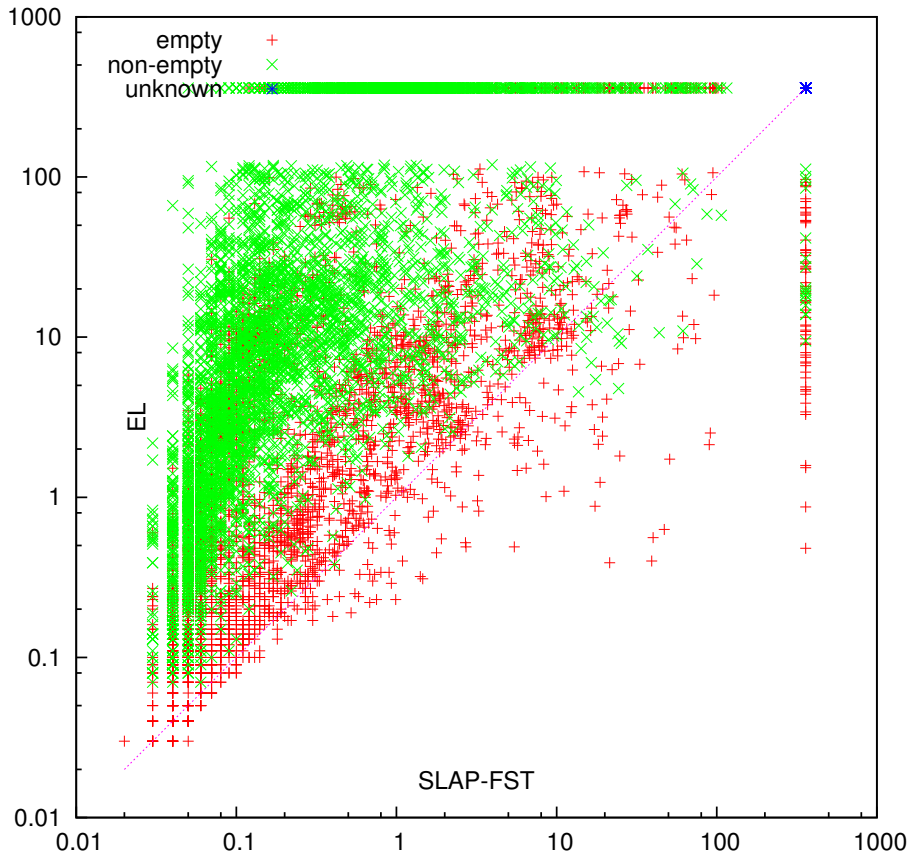
Runtime in seconds. Timeout at 120s.

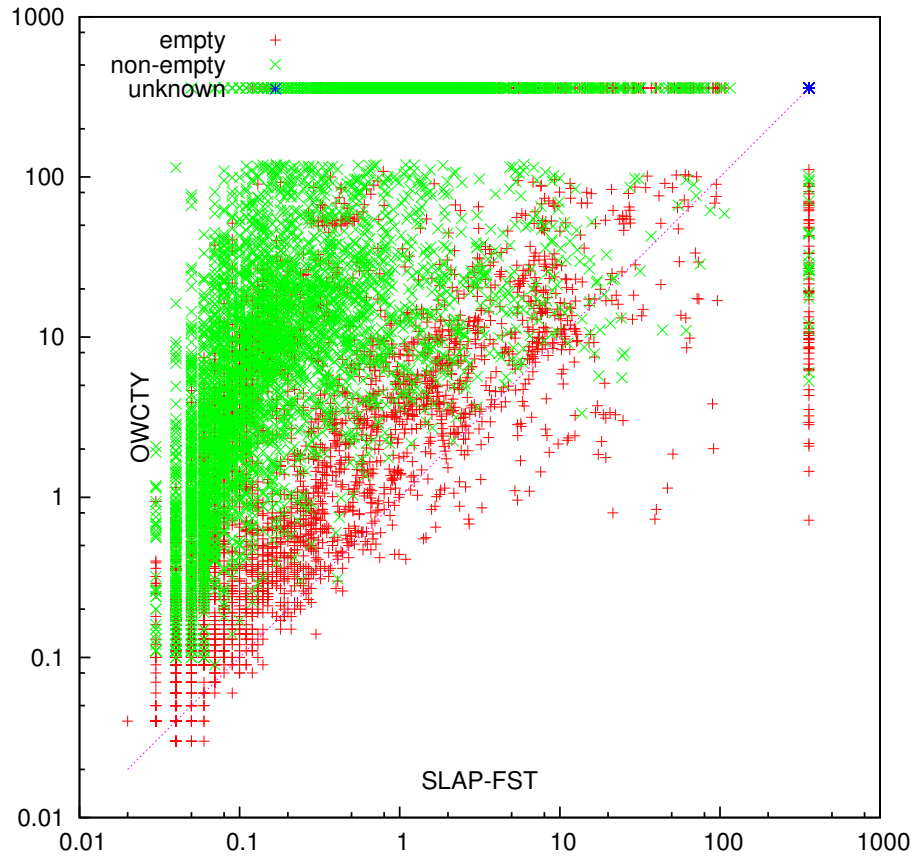




Scatter Plots: SLAP-FST vs. Fully Symbolic

Runtime in seconds. Timeout at 120s.





Conclusion

- SLAP, a new hybrid approach:
  - Aggregates states at the product level, not in the Kripke structure
  - Can therefore adjust dynamically to the features (self-loops and acceptance conditions) present in the property automaton.
  - SLAP-FST: a variant using a fully-symbolic algorithm for terminal states.
- SLAP-FST most competitive algorithm of those we compared, **on this benchmark**
- On-the-fly computations have strong impact when counterexamples exist. Hybrid approaches outperform fully symbolic approaches in these cases.

`http://move.lip6.fr/software/DDD/ltl_bench.html`