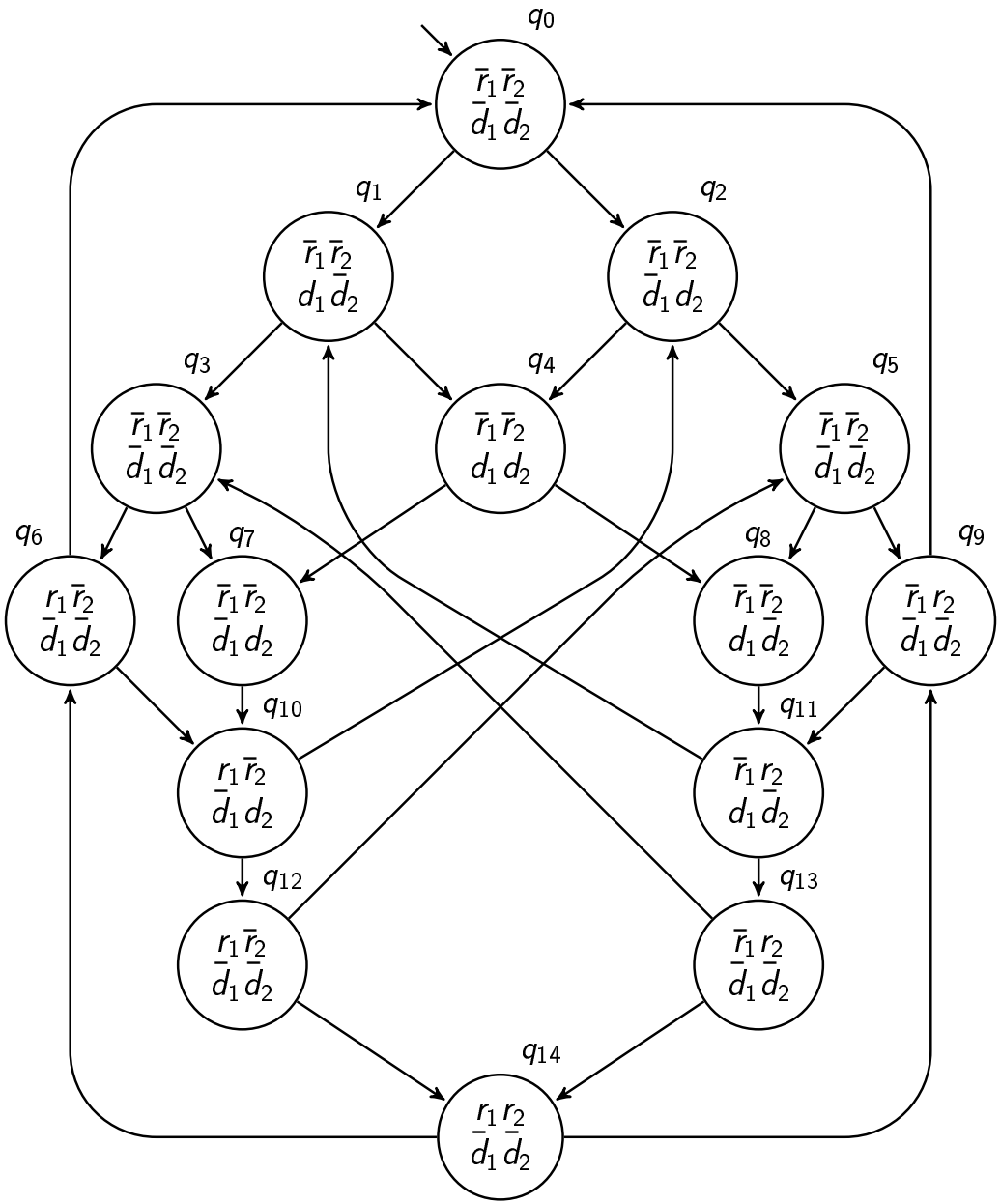
Emptiness checks and Partial Orders

Alexandre Duret-Lutz

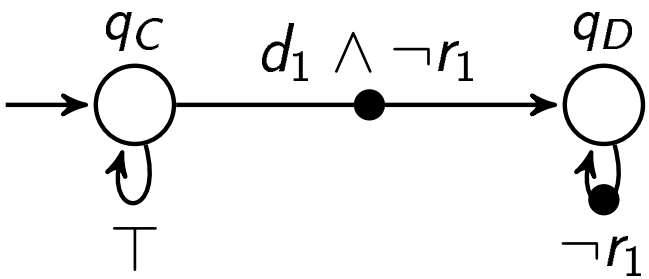
mars 2009

Kripke structure



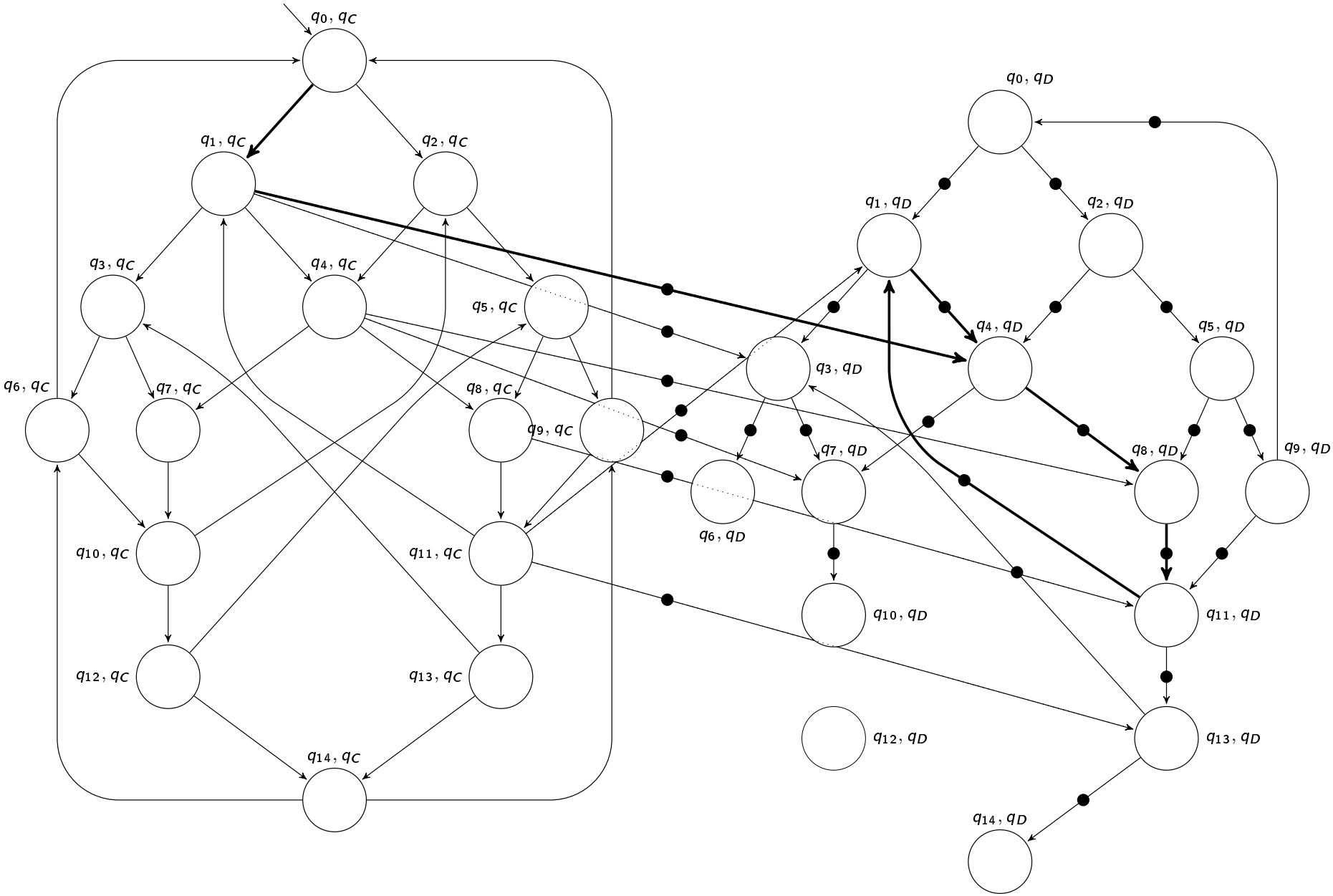
Formula to verify

$\mathbf{G}(d_1 \rightarrow \mathbf{F}\, r_1)$.



$A_{\neg\, \mathbf{G}(d_1 \rightarrow \mathbf{F}\, r_1)}$

Synchronized product



Today's special

1. On-the-Fly Emptiness Checks for Generalized Büchi Automata
2. Using Partial Orders to Reduce the State Space

Part I

On-the-Fly Emptiness Checks for GBA

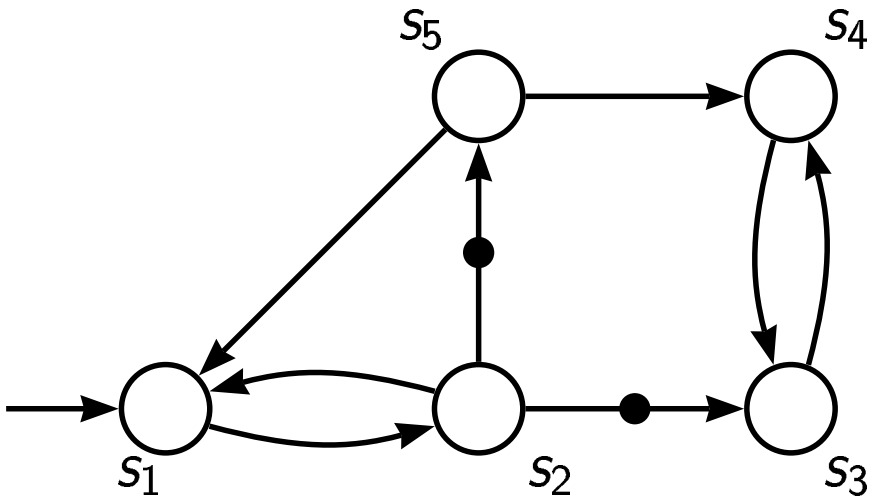
Jean-Michel Couvreur, Alexandre Duret-Lutz, Denis Poitrenaud
12th International SPIN Workshop on Model Checking of Software,
August 2005, San Francisco.

Büchi Automata

A (transition-based) Büchi automaton has:

- A set of states, with a designated initial state,
- A set of transitions between states,
- A set of accepting transitions.

An infinite run of this automaton is accepting if it visits an accepting transition infinitely often.

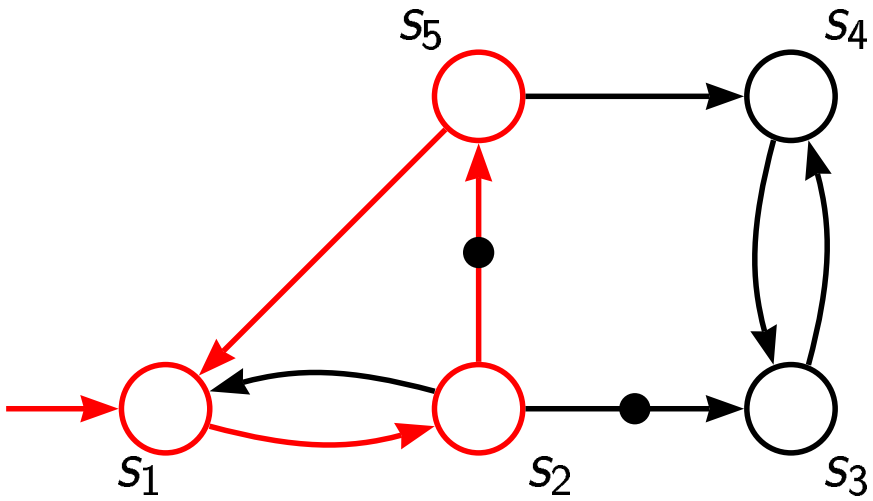


Büchi Automata

A (transition-based) Büchi automaton has:

- A set of states, with a designated initial state,
- A set of transitions between states,
- A set of accepting transitions.

An infinite run of this automaton is accepting if it visits an accepting transition infinitely often.



Emptiness Check

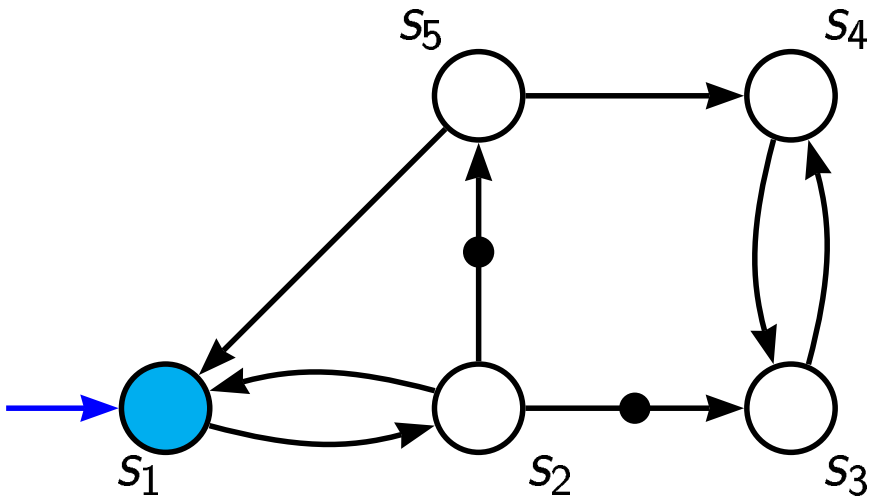
Emptiness Check = Does an automaton have no accepting run?

$\Longrightarrow$ Search for an accepting cycle reachable from the initial state.

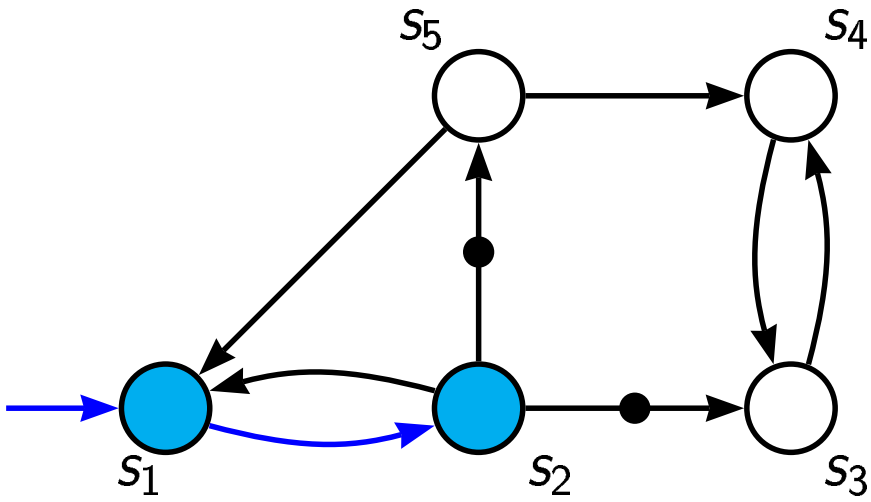
Emptiness Checks History

nested DFS	Courcoubetis et al.	'90
	Godefroid & Holzmann	'93
	Holzmann et al.	'96
	Gastin et al.	'04
	Schwoon & Esparza	'04

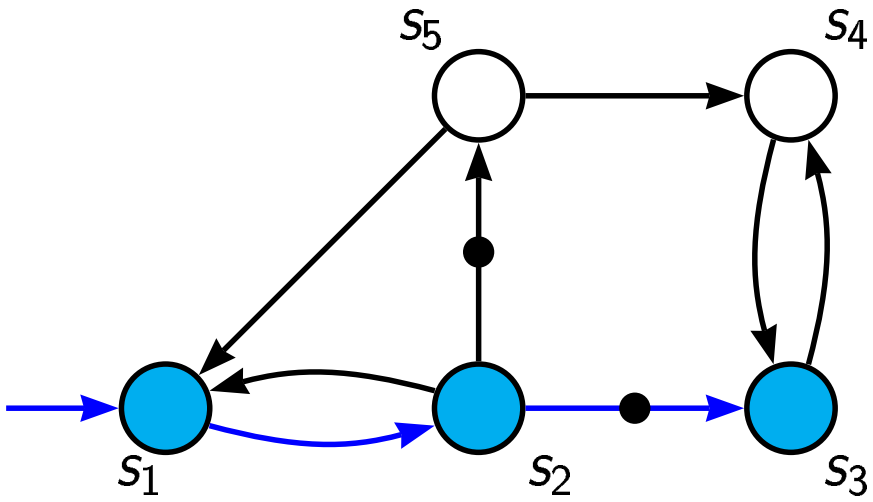
Nested DFS



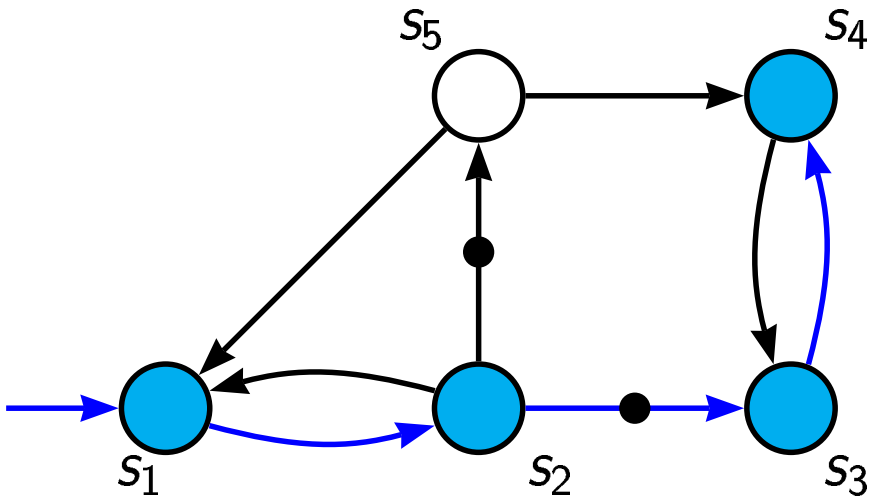
Nested DFS



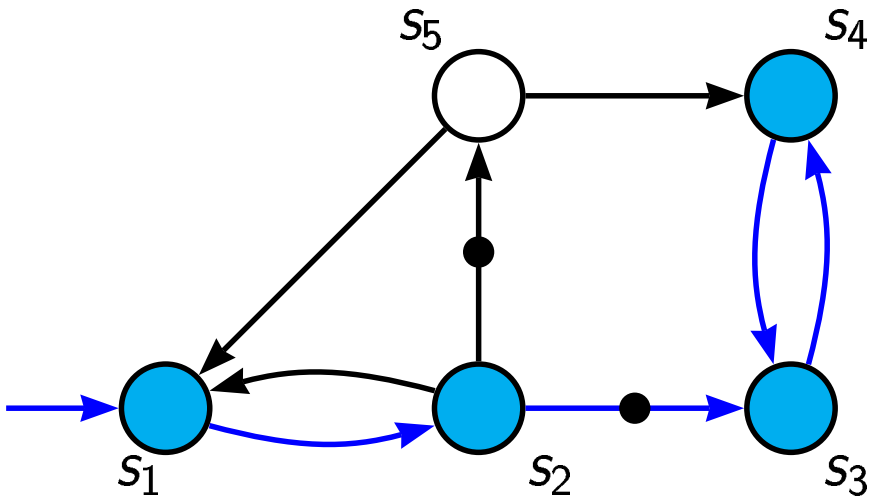
Nested DFS



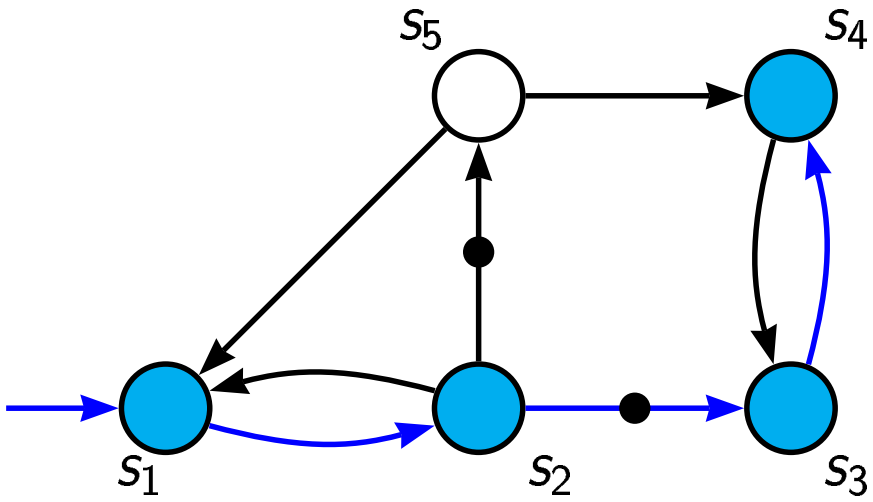
Nested DFS



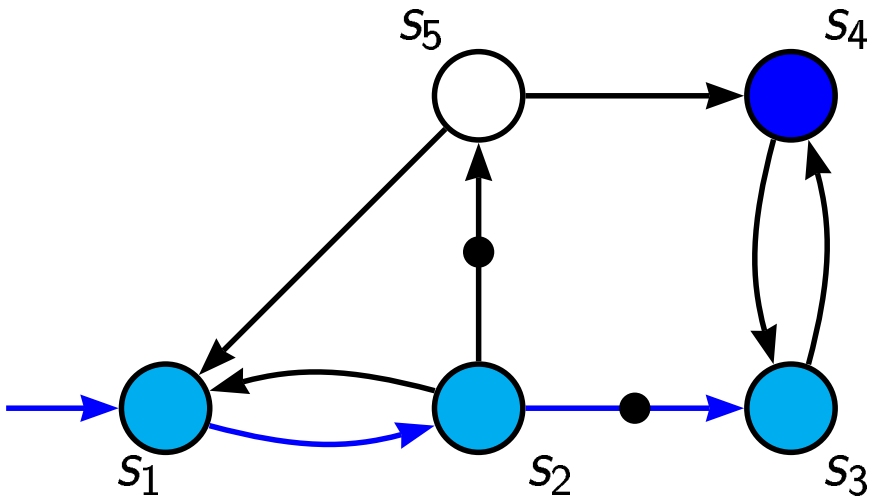
Nested DFS



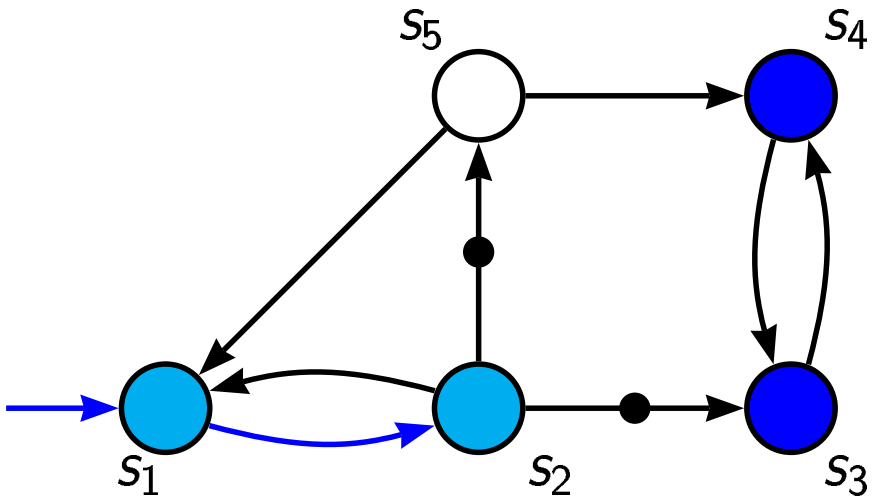
Nested DFS



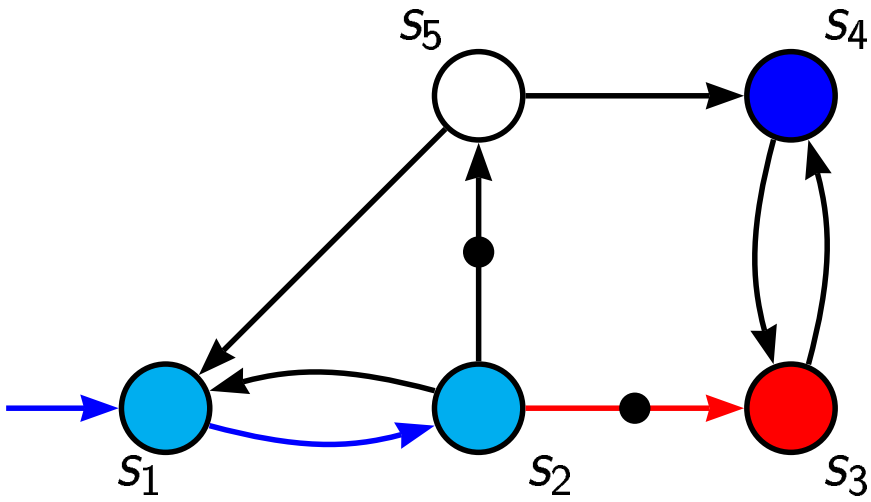
Nested DFS



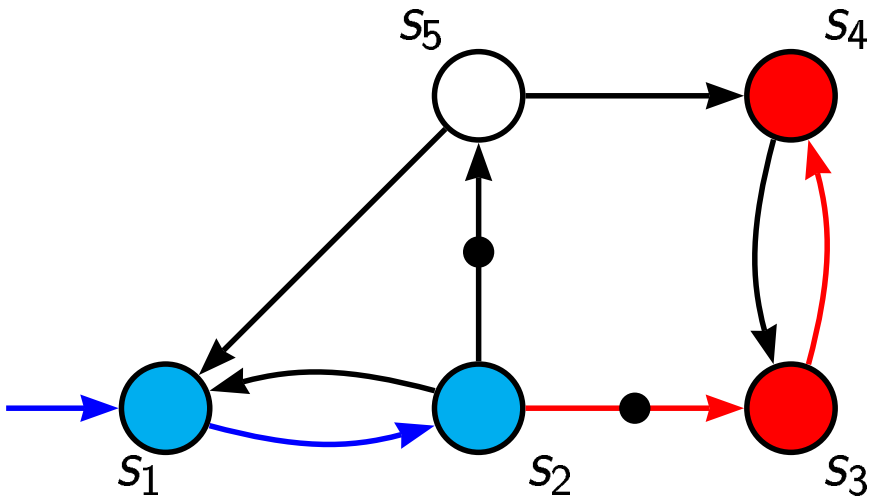
Nested DFS



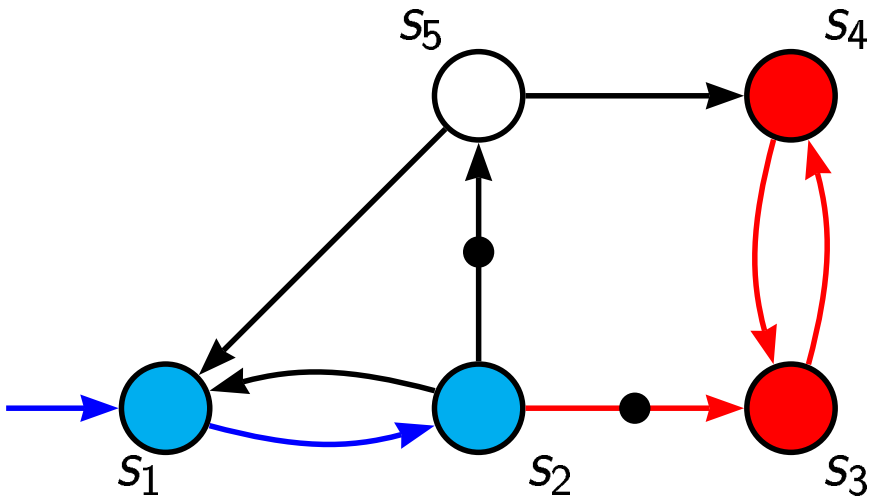
Nested DFS



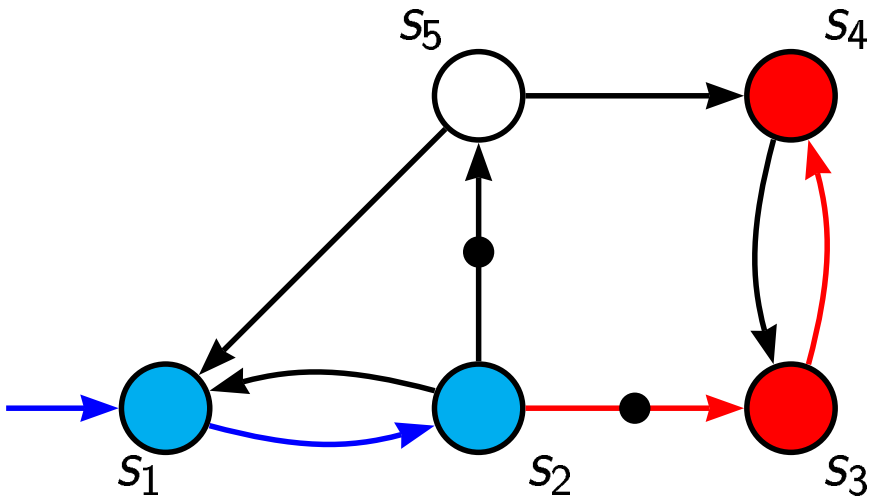
Nested DFS



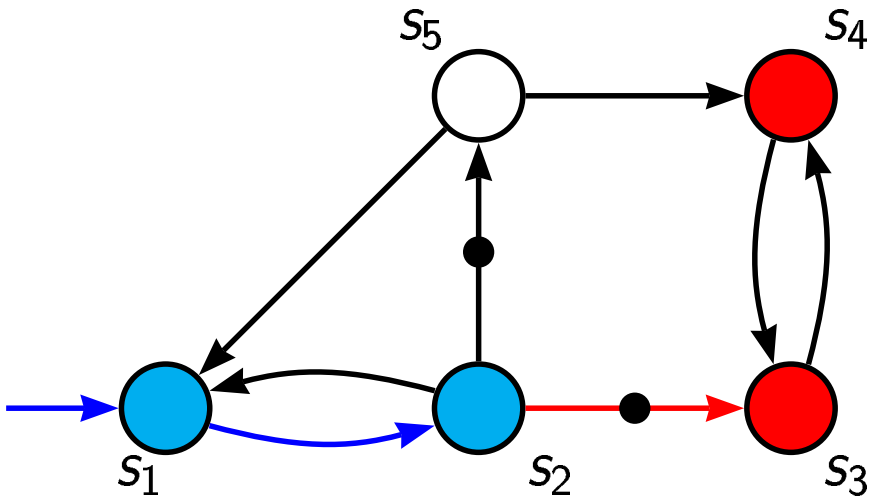
Nested DFS



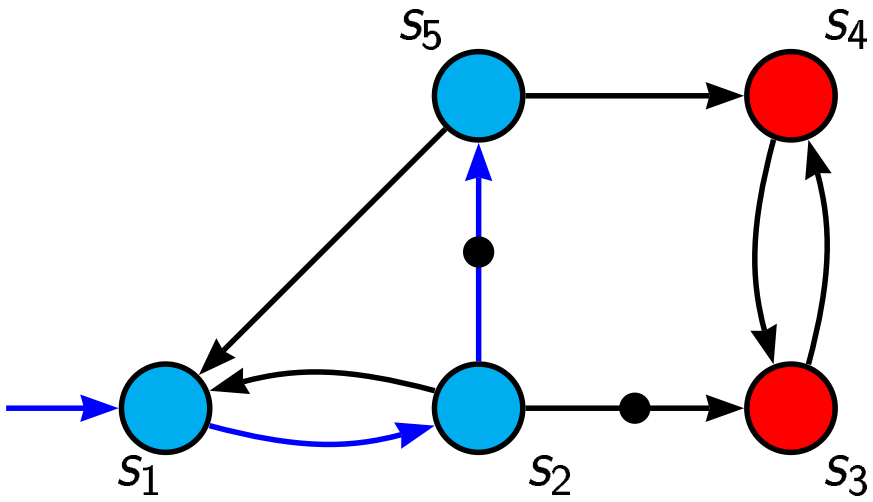
Nested DFS



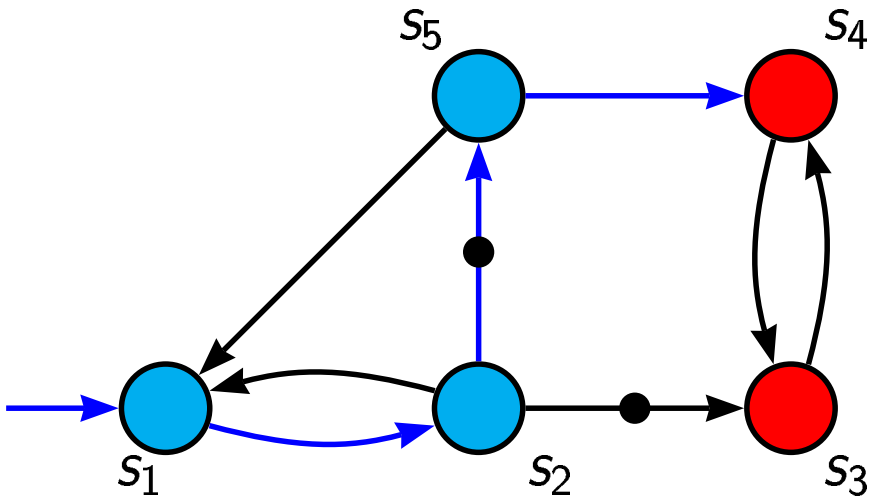
Nested DFS



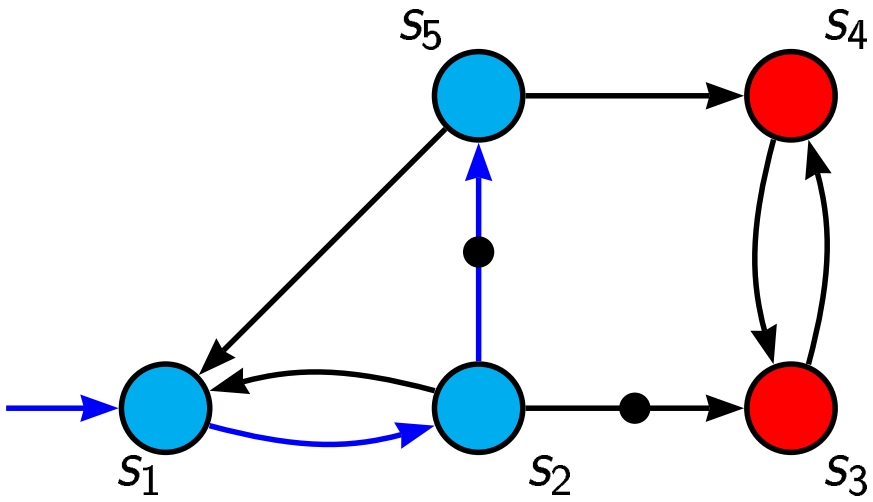
Nested DFS



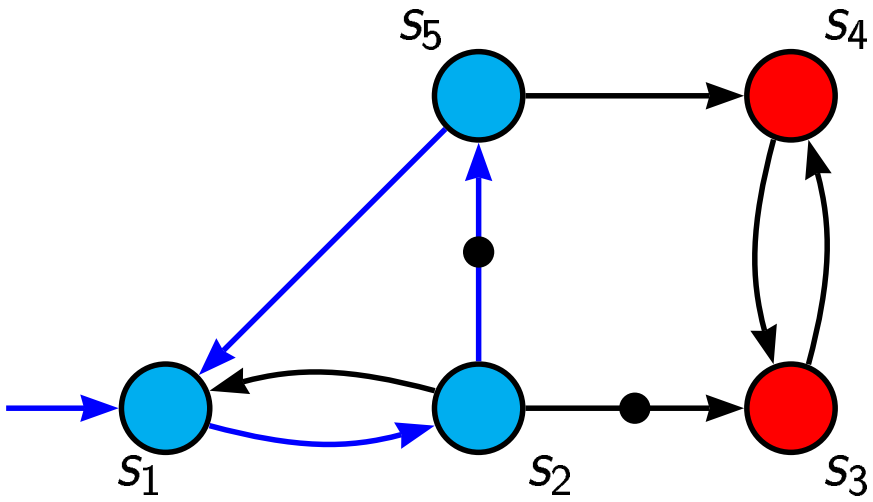
Nested DFS



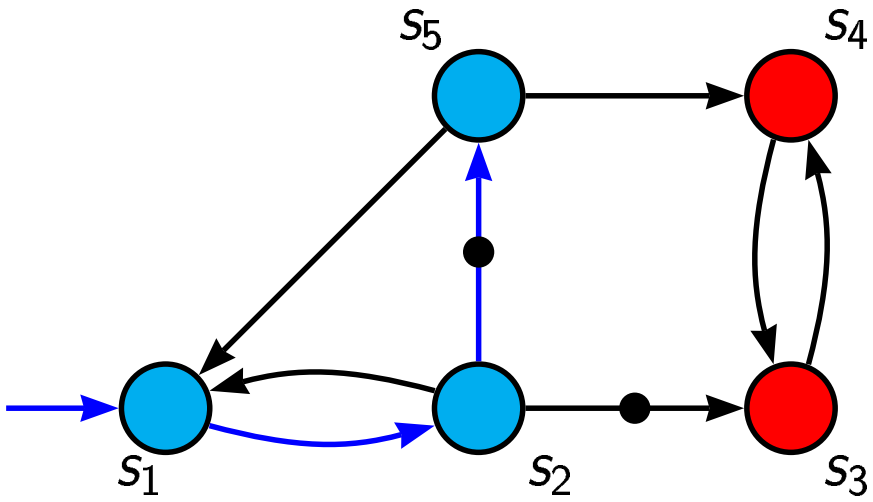
Nested DFS



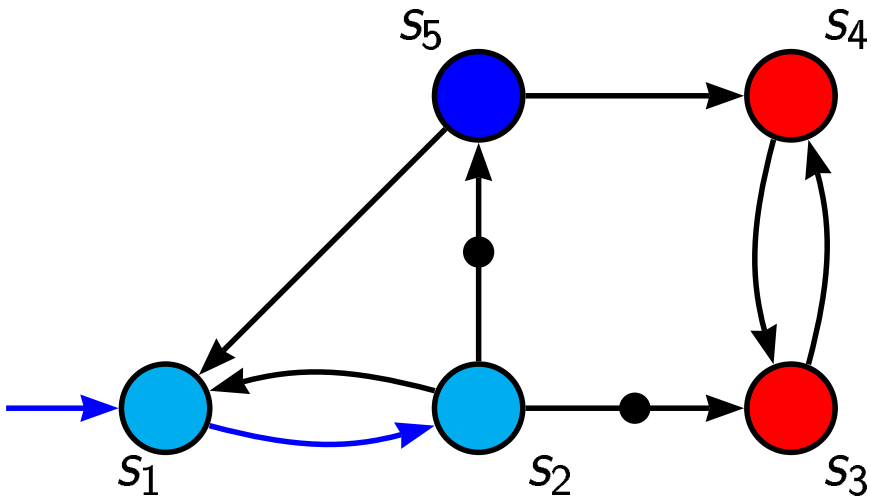
Nested DFS



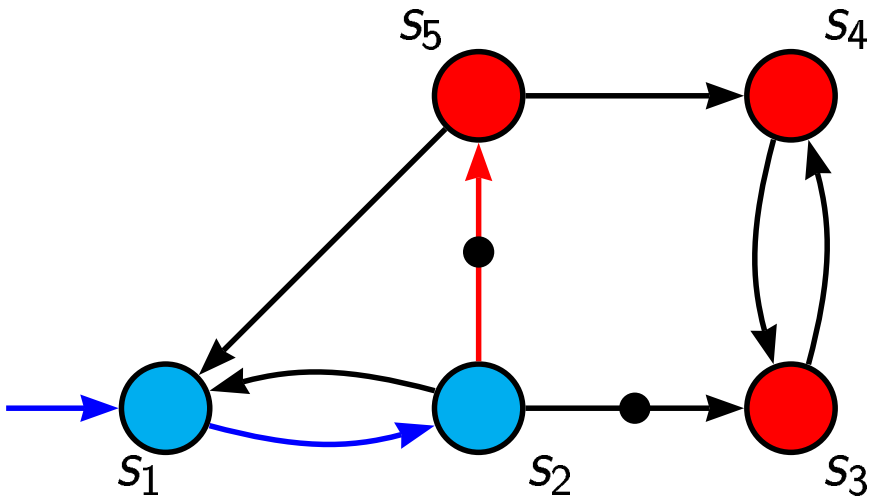
Nested DFS



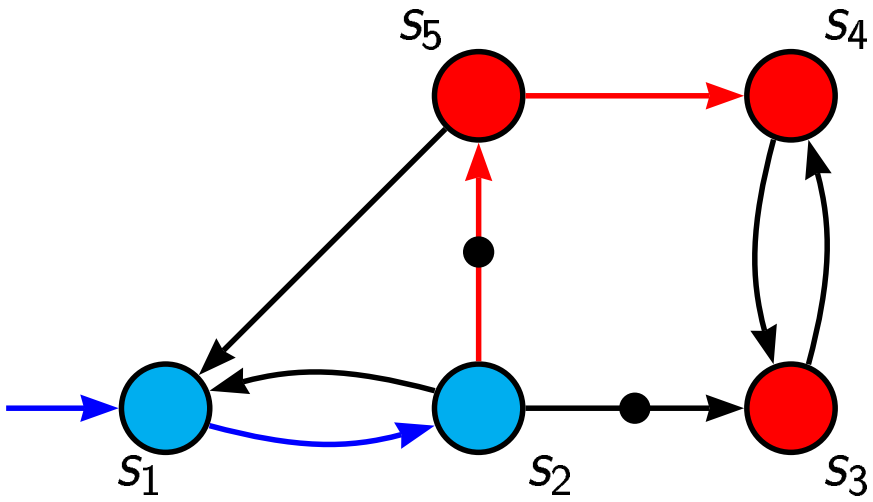
Nested DFS



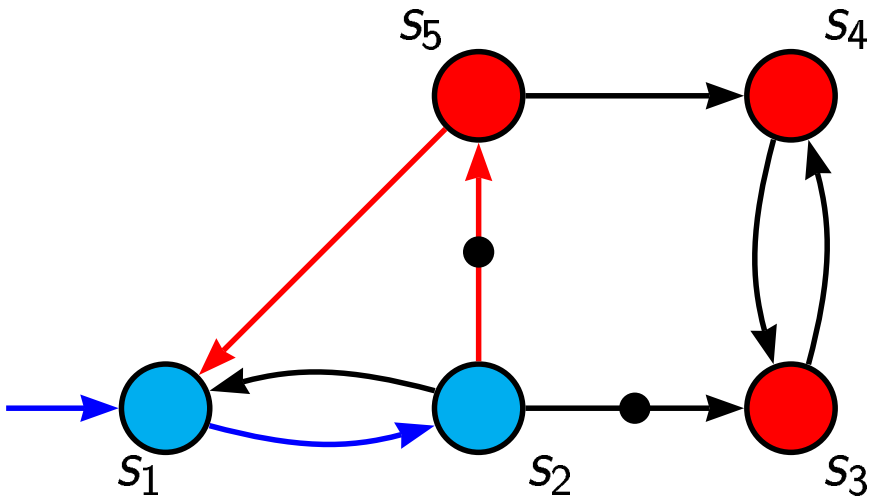
Nested DFS



Nested DFS

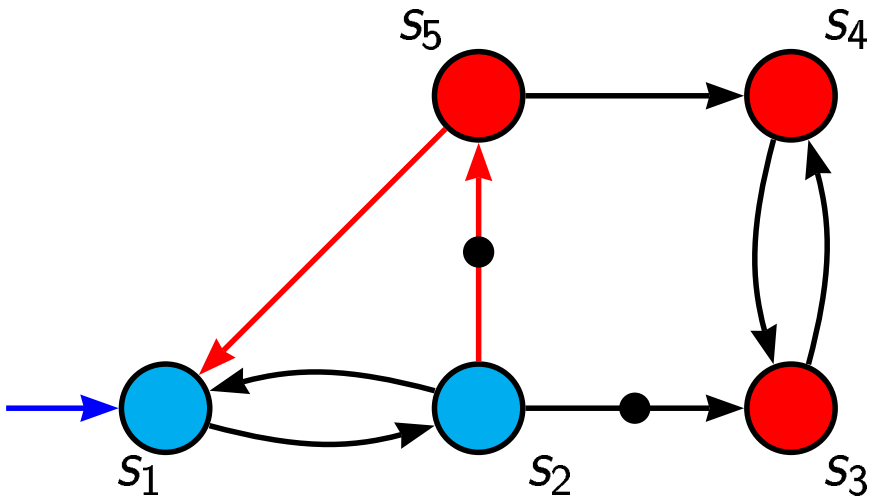


Nested DFS



Found!

Nested DFS



Found!

	entries in hash table	hash table size in bits	search stack depth	states traversed
upper bounds:	n	$n(s+2)$	n	$2n$

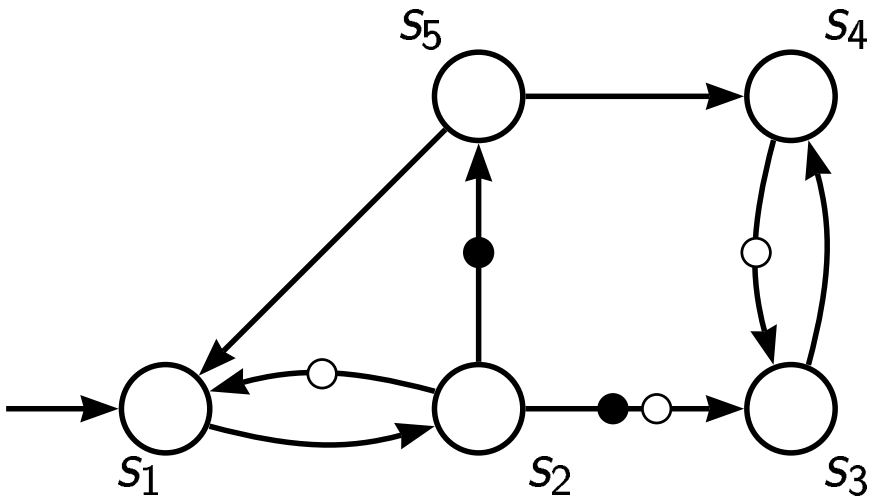
n = number of states; s = bits per state.

Generalized Büchi Automata

A Generalized (transition-based) Büchi automaton has:

- A set of states, with a designated initial state,
- A set of transitions between states,
- A set of accepting sets of transitions.

An infinite run of this automaton is accepting if it visits a transition from each accepting set infinitely often.

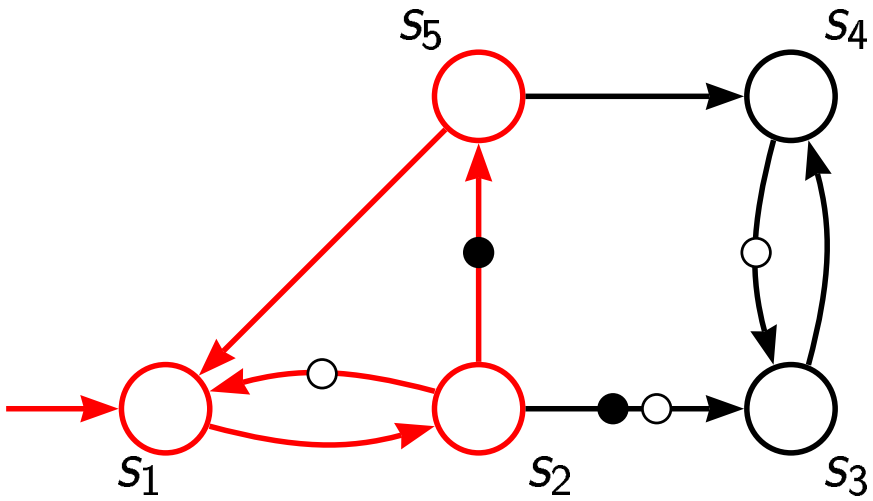


Generalized Büchi Automata

A Generalized (transition-based) Büchi automaton has:

- A set of states, with a designated initial state,
- A set of transitions between states,
- A set of accepting sets of transitions.

An infinite run of this automaton is accepting if it visits a transition from each accepting set infinitely often.



Degeneralization

A generalized automaton with

- n states
- m acceptance conditions

can be degeneralized into an automaton with

- nm states at worst
- 1 acceptance condition

Nested DFS on Generalized Büchi Automata

entries in hash table	hash table size in bits	search stack depth	states traversed
n	$n \quad (s \quad + 2)$	n	$2n$

n states,
s bits per state.

Nested DFS on Generalized Büchi Automata

entries in hash table	hash table size in bits	search stack depth	states traversed
nm	$nm(s_d + 2)$	nm	$2nm$

n states, m acceptance conditions,
s_d bits per degeneralized state.

Nested DFS on Generalized Büchi Automata

entries in hash table	hash table size in bits	search stack depth	states traversed
nm	$nm(s_d + 2)$	nm	$2nm$
n	$n(s_g + 2m)$	nm	$2nm$

n states, m acceptance conditions,
s_d bits per degeneralized state, s_g bits per generalized state ($s_g \leq s_d$).

Emptiness Checks History

nested DFS	Courcoubetis et al.	'90
	Godefroid & Holzmann	'93
	Holzmann et al.	'96
	Gastin et al.	'04
	Schwoon & Esparza	'04

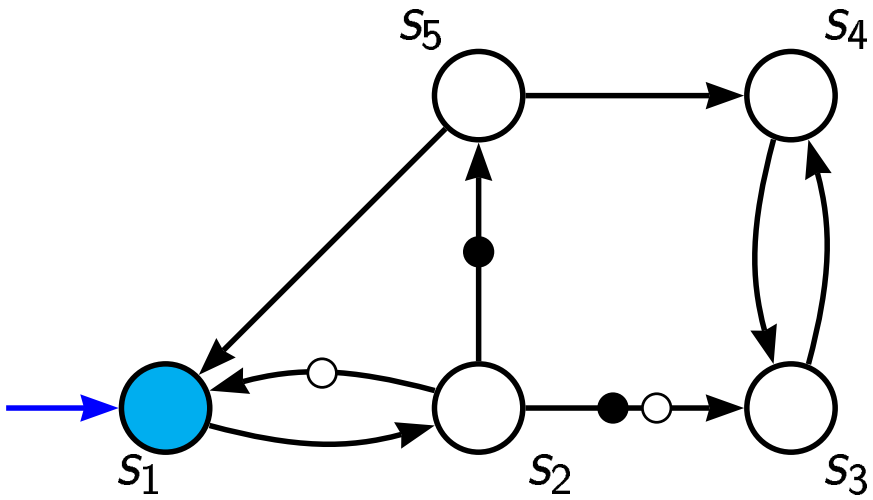
Emptiness Checks History

	degeneralized	generalized
nested DFS	Courcoubetis et al. '90	
	Godefroid & Holzmann '93	
	Holzmann et al. '96	
	Gastin et al. '04	
	Schwoon & Esparza '04	

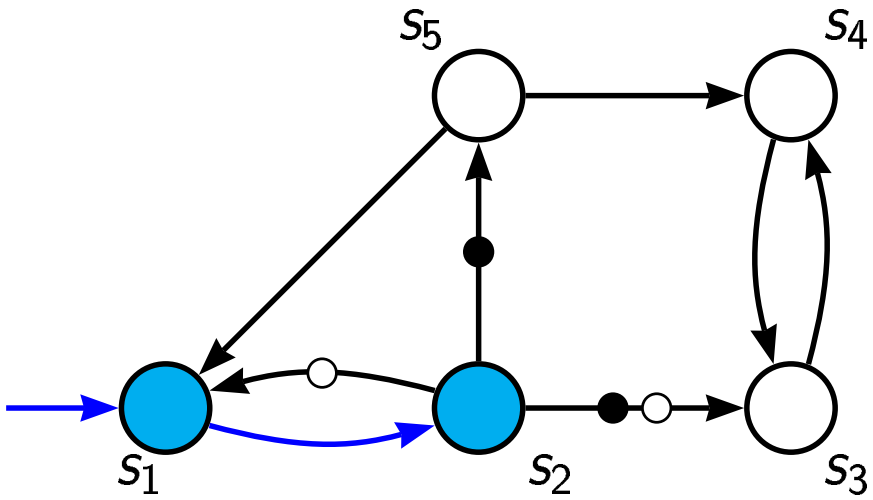
Emptiness Checks History

	degeneralized		generalized	
nested DFS	Courcoubetis et al.	'90		
	Godefroid & Holzmann	'93		
	Holzmann et al.	'96	Tauriainen	'03
	Gastin et al.	'04		
	Schwoon & Esparza	'04		

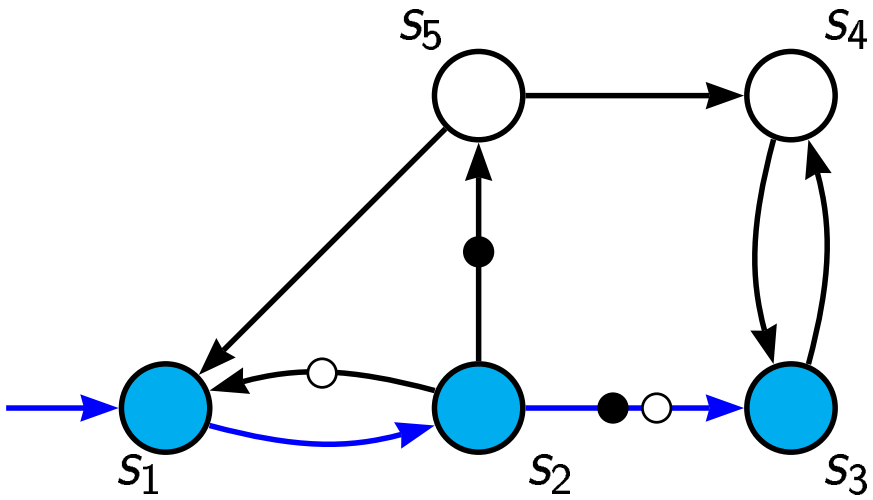
Generalized Nested DFS (Tauriainen'03)



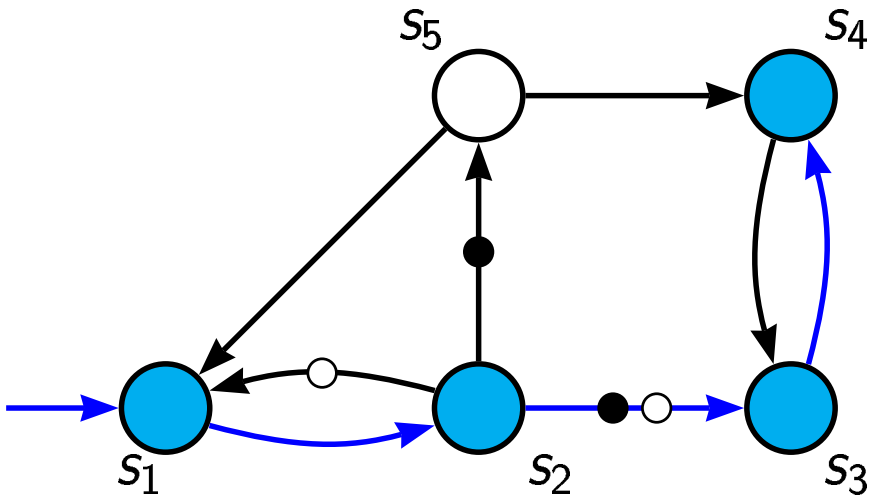
Generalized Nested DFS (Tauriainen'03)



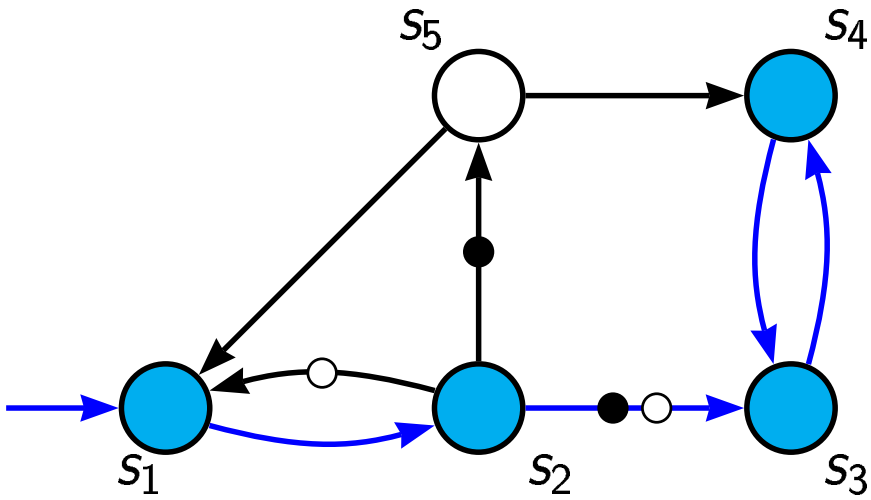
Generalized Nested DFS (Tauriainen'03)



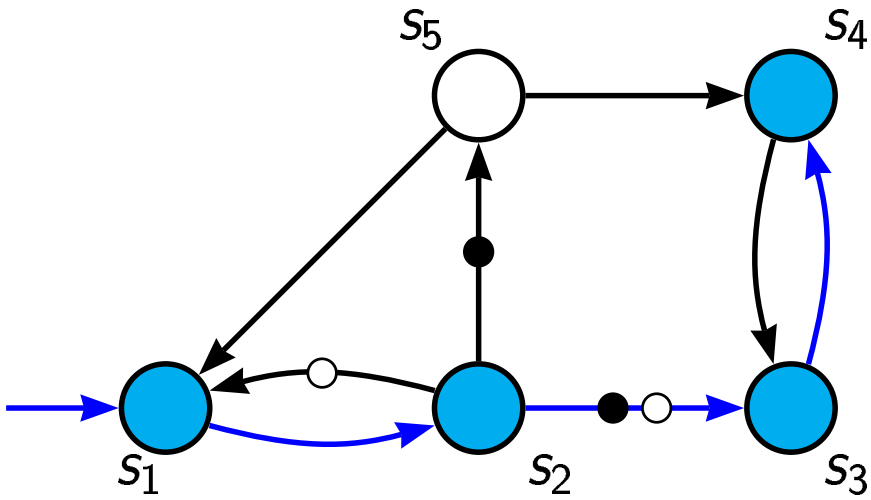
Generalized Nested DFS (Tauriainen'03)



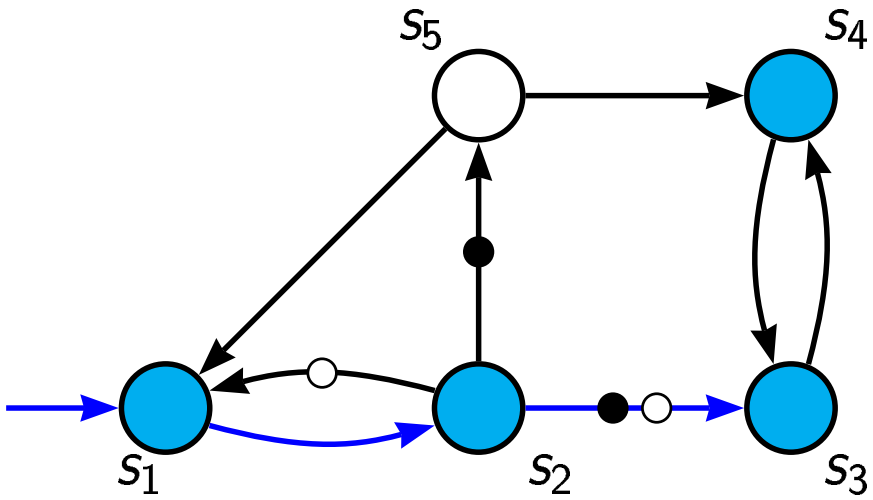
Generalized Nested DFS (Tauriainen'03)



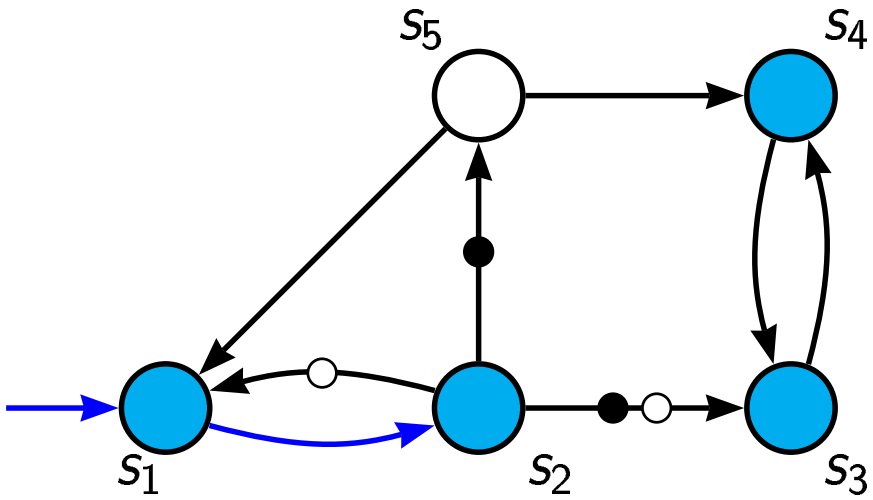
Generalized Nested DFS (Tauriainen'03)



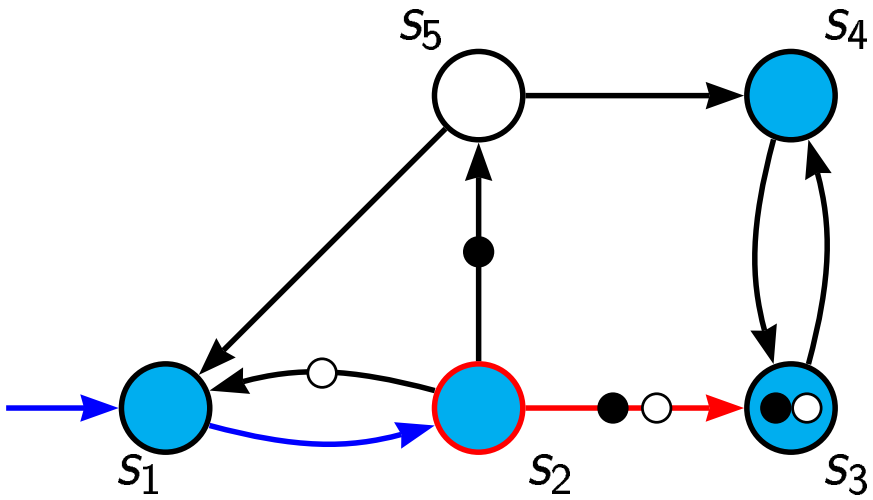
Generalized Nested DFS (Tauriainen'03)



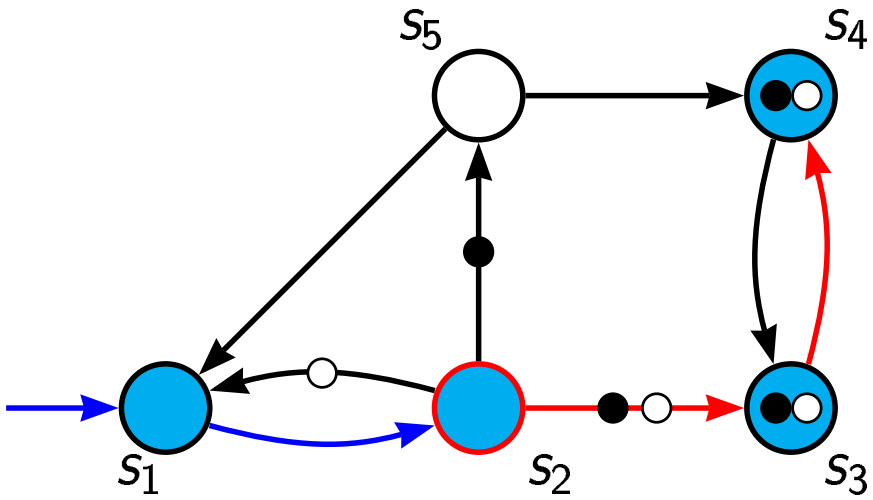
Generalized Nested DFS (Tauriainen'03)



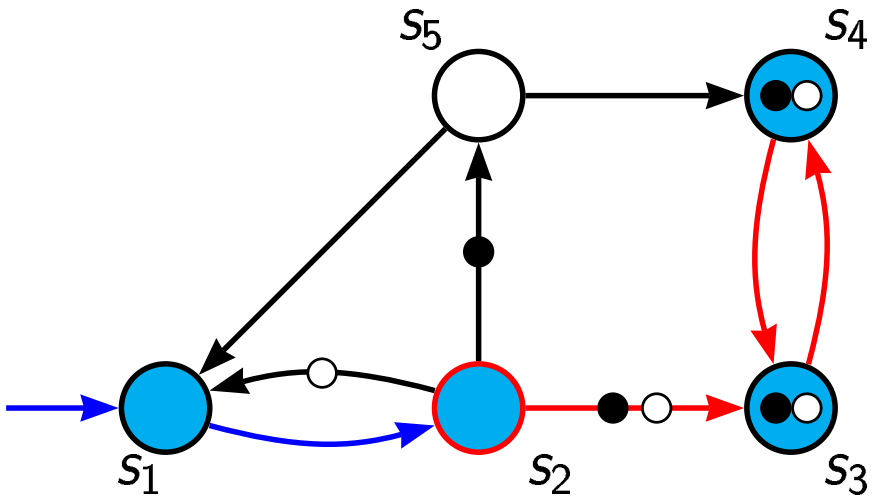
Generalized Nested DFS (Tauriainen'03)



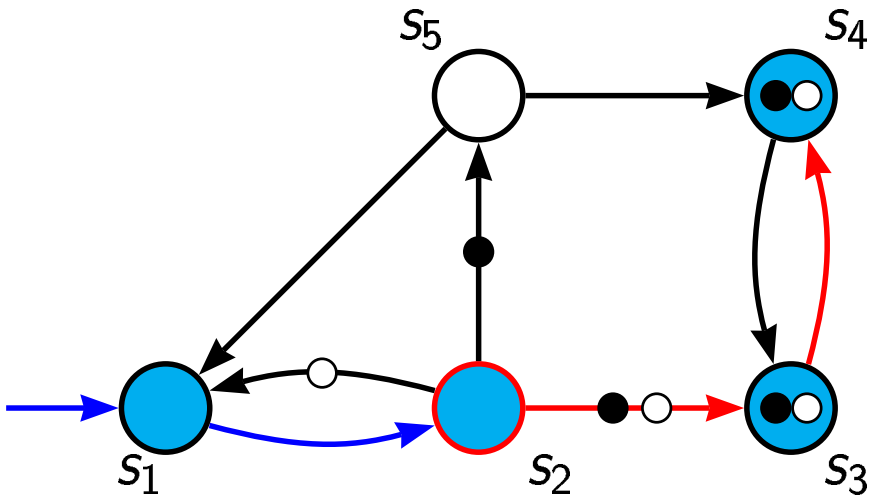
Generalized Nested DFS (Tauriainen'03)



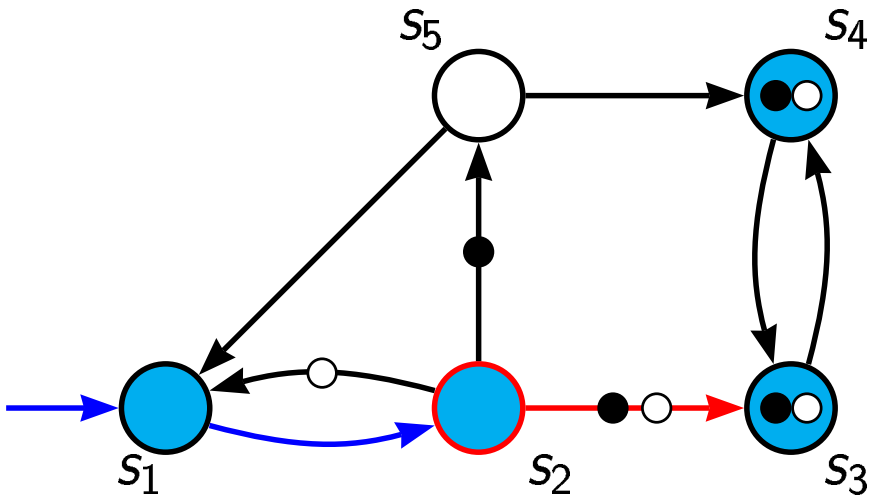
Generalized Nested DFS (Tauriainen'03)



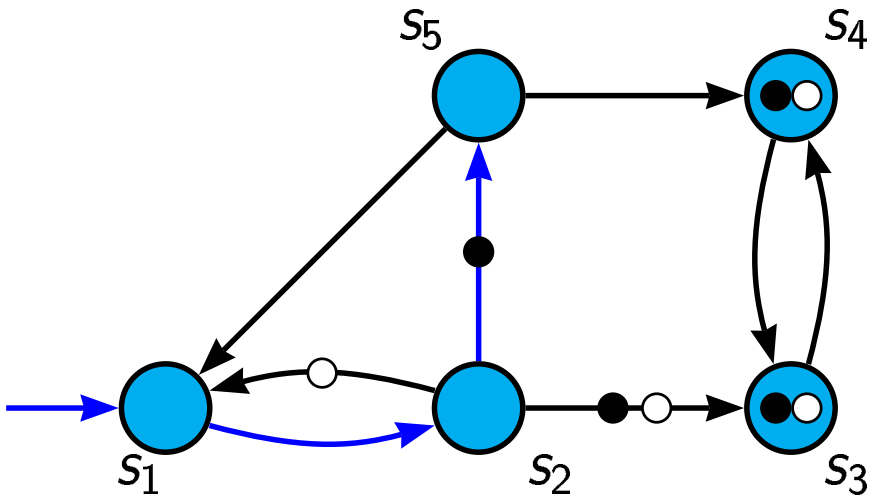
Generalized Nested DFS (Tauriainen'03)



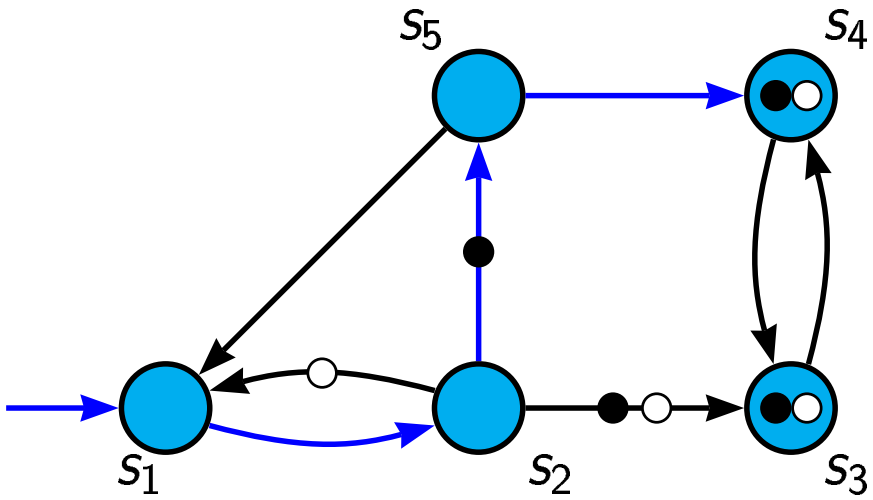
Generalized Nested DFS (Tauriainen'03)



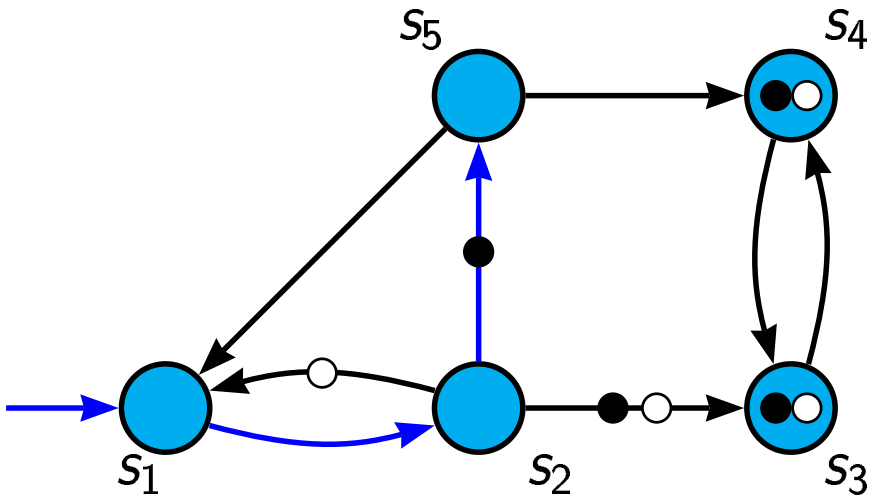
Generalized Nested DFS (Tauriainen'03)



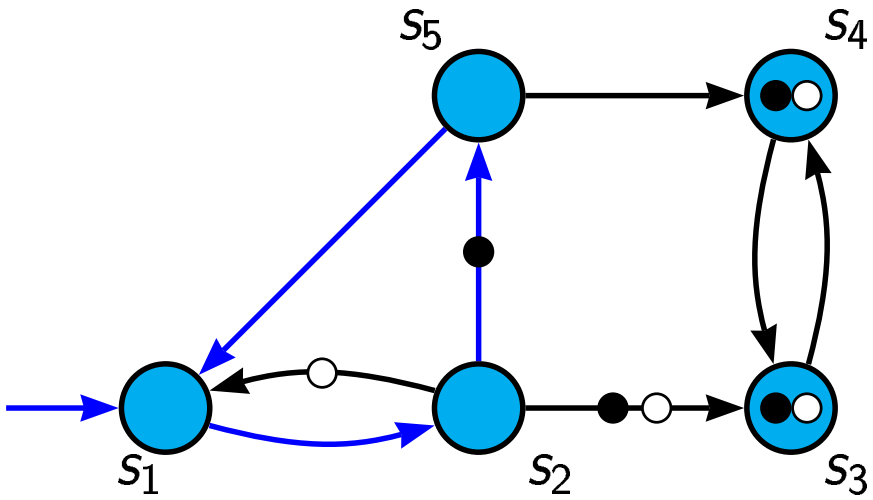
Generalized Nested DFS (Tauriainen'03)



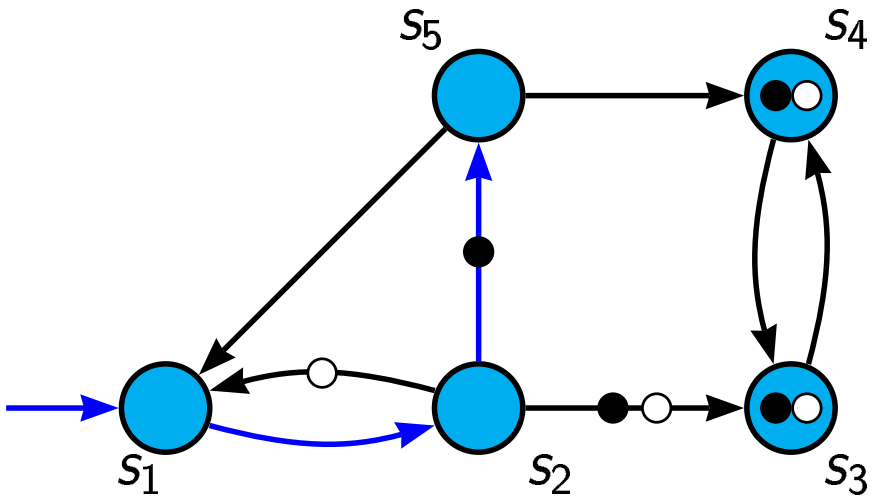
Generalized Nested DFS (Tauriainen'03)



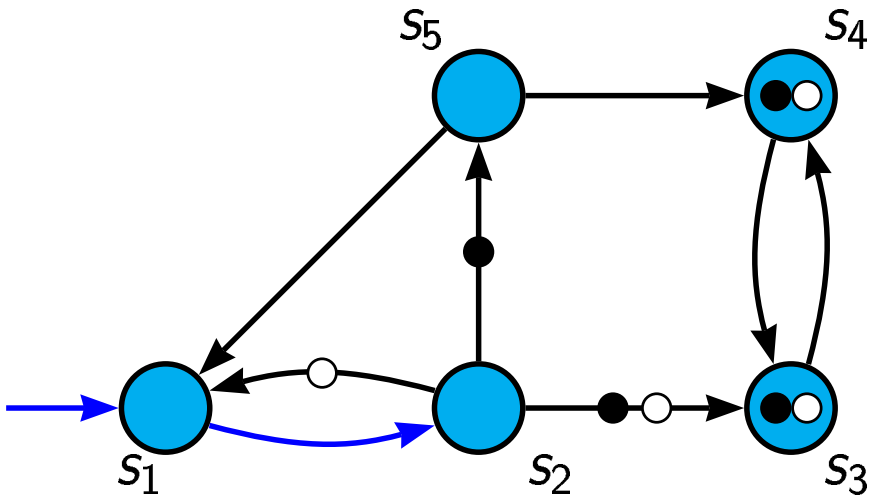
Generalized Nested DFS (Tauriainen'03)



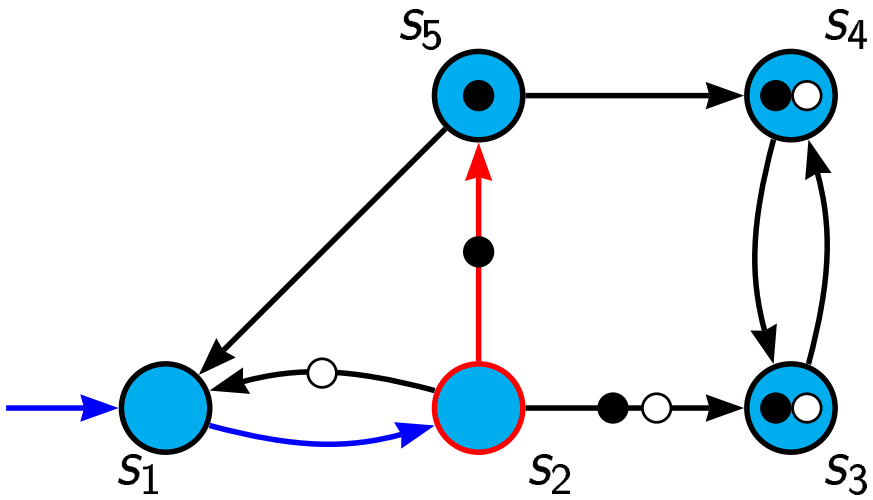
Generalized Nested DFS (Tauriainen'03)



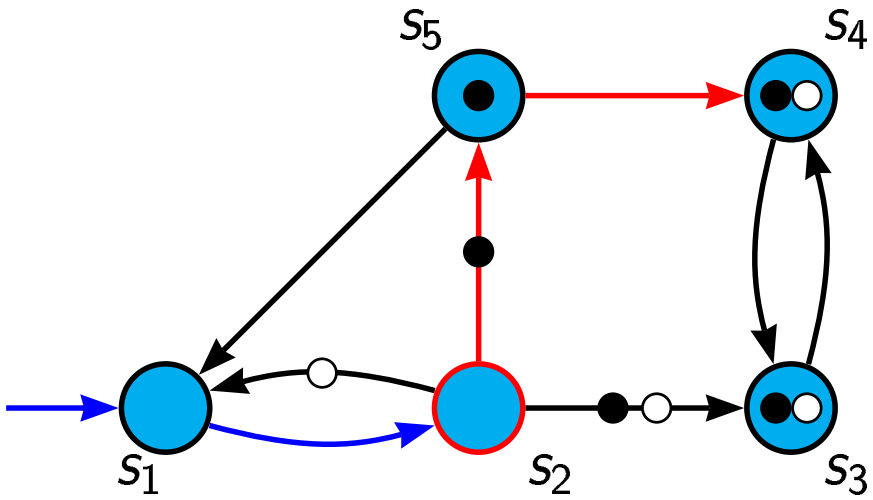
Generalized Nested DFS (Tauriainen'03)



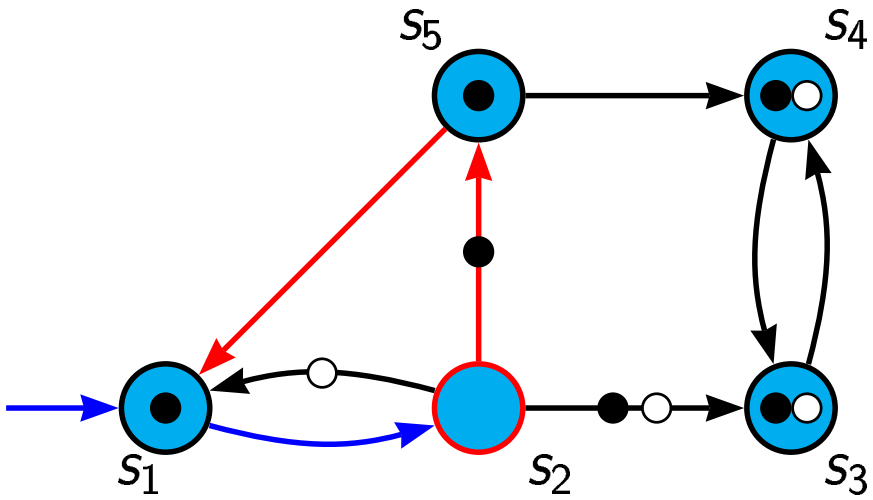
Generalized Nested DFS (Tauriainen'03)



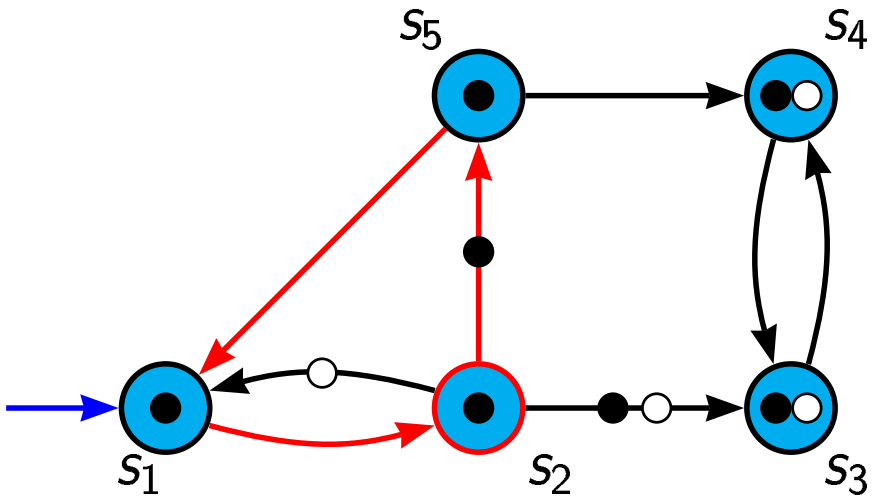
Generalized Nested DFS (Tauriainen'03)



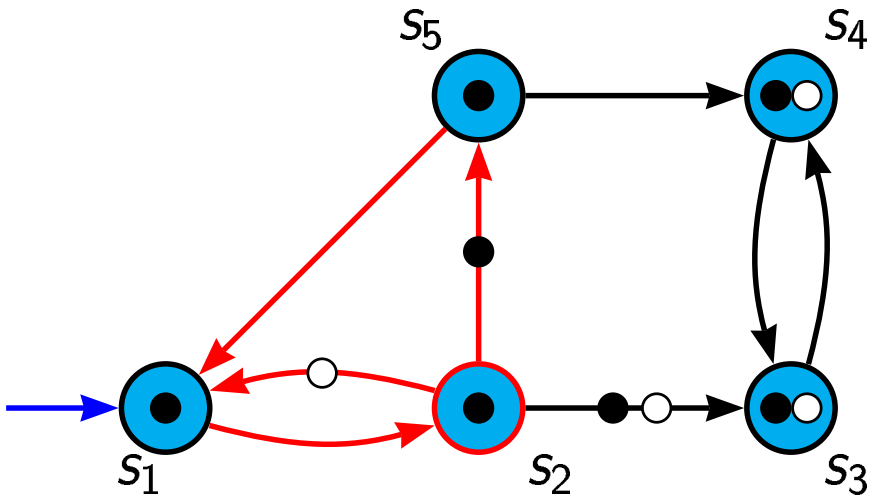
Generalized Nested DFS (Tauriainen'03)



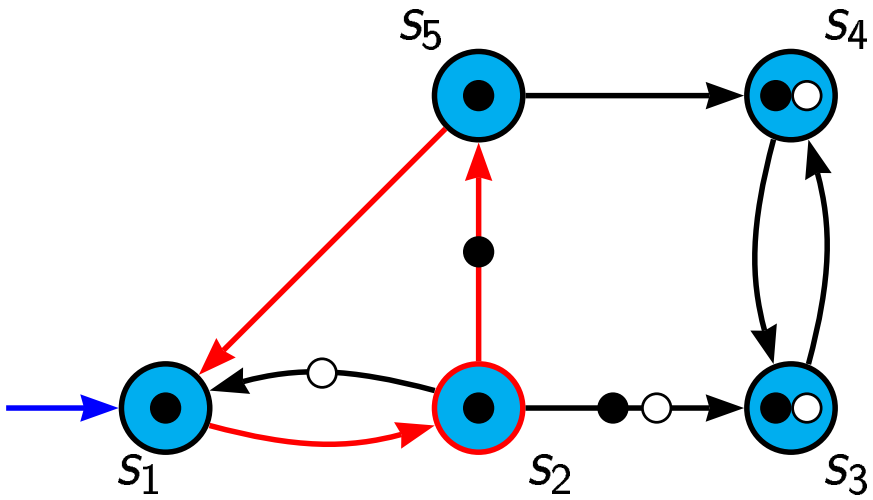
Generalized Nested DFS (Tauriainen'03)



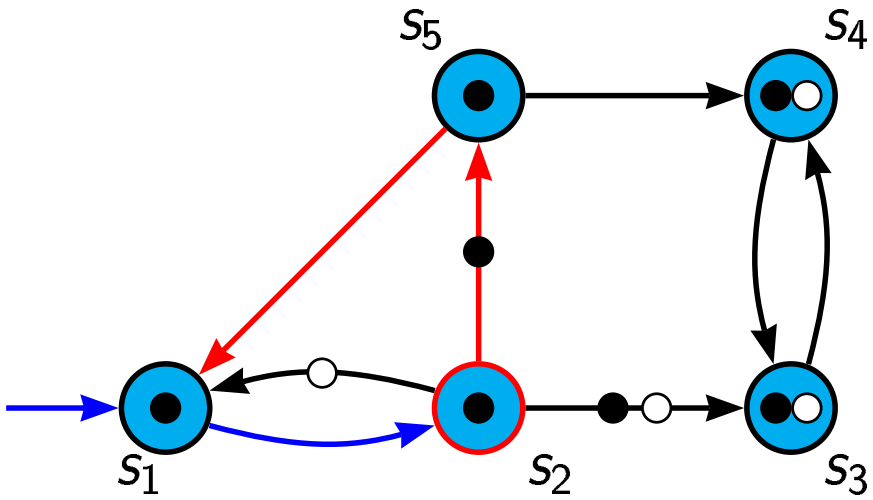
Generalized Nested DFS (Tauriainen'03)



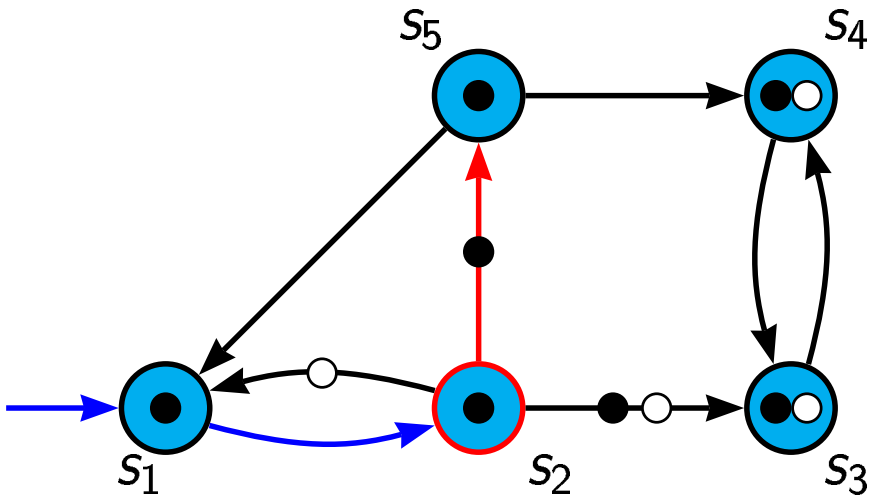
Generalized Nested DFS (Tauriainen'03)



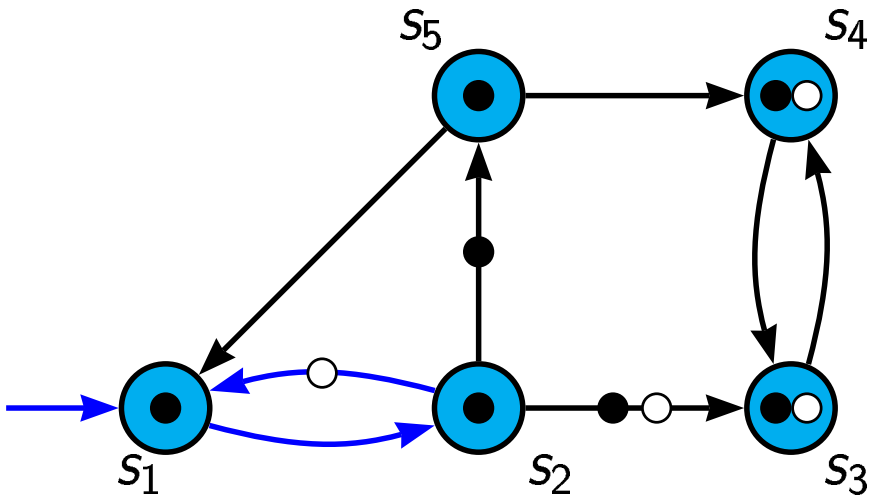
Generalized Nested DFS (Tauriainen'03)



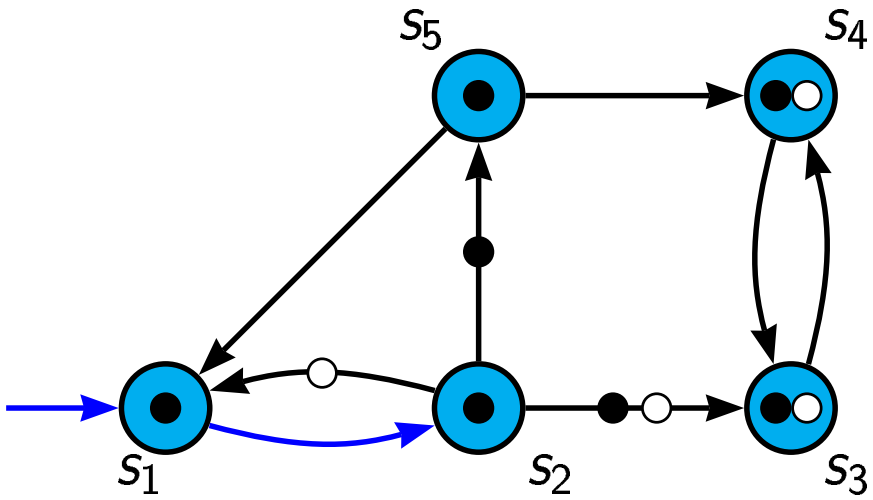
Generalized Nested DFS (Tauriainen'03)



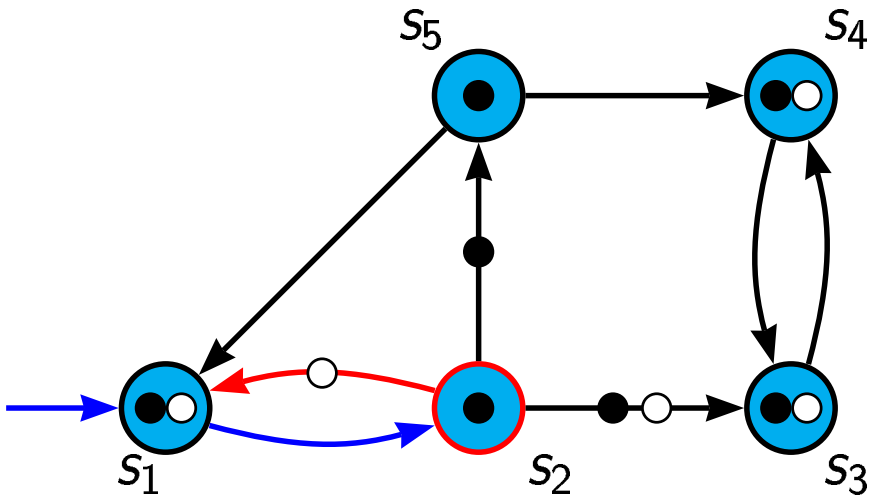
Generalized Nested DFS (Tauriainen'03)



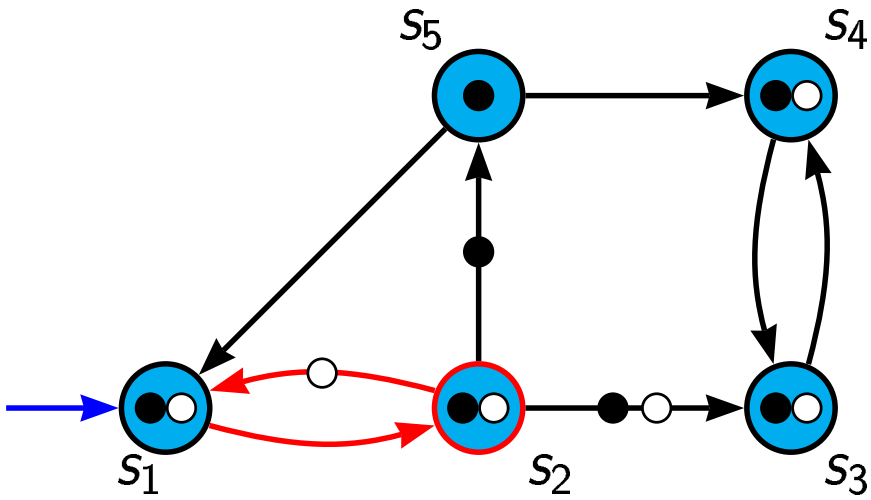
Generalized Nested DFS (Tauriainen'03)



Generalized Nested DFS (Tauriainen'03)

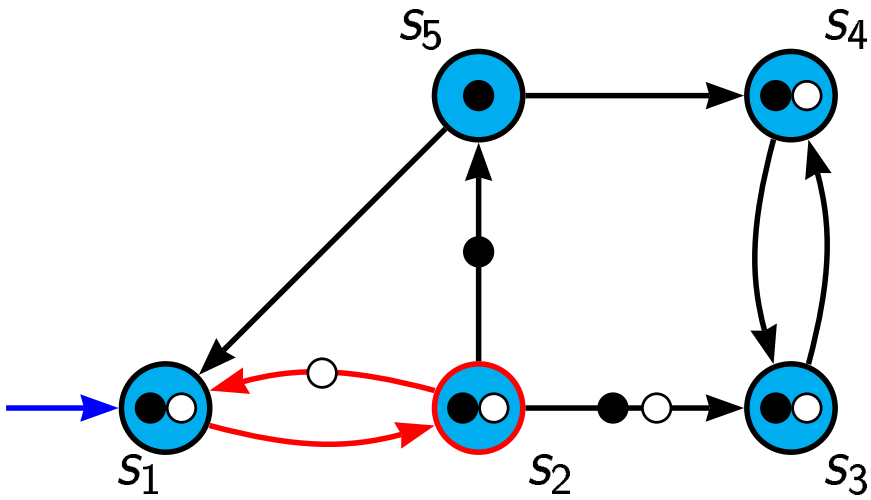


Generalized Nested DFS (Tauriainen'03)



Found!

Generalized Nested DFS (Tauriainen'03)



Found!

	entries in hash table	hash table size in bits	search stack depth	states traversed
degen+NDFS	n	$n(s_g + 2m)$	nm	$2nm$
gen. NDFS	n	$n(s_g + m)$	$2n$	$n(m+1)$

Emptiness Checks History

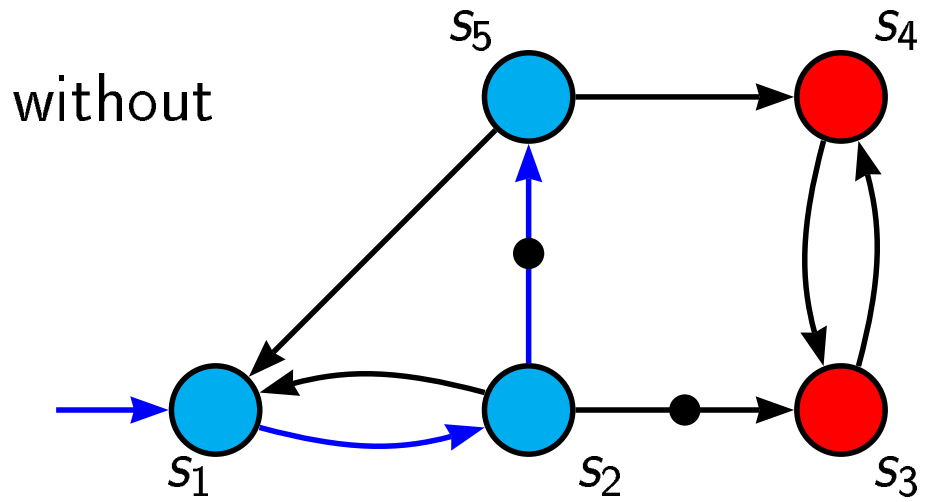
	degeneralized		generalized	
nested DFS	Courcoubetis et al.	'90		
	Godefroid & Holzmann	'93		
	Holzmann et al.	'96	Tauriainen	'03
	Gastin et al.	'04		
	Schwoon & Esparza	'04		

Emptiness Checks History

	degeneralized		generalized	
nested DFS	Courcoubetis et al.	'90		
	Godefroid & Holzmann	'93		
	Holzmann et al.	'96	Tauriainen	'03
	Gastin et al.	'04		
	Schwoon & Esparza	'04	Couvreur et al.	'05

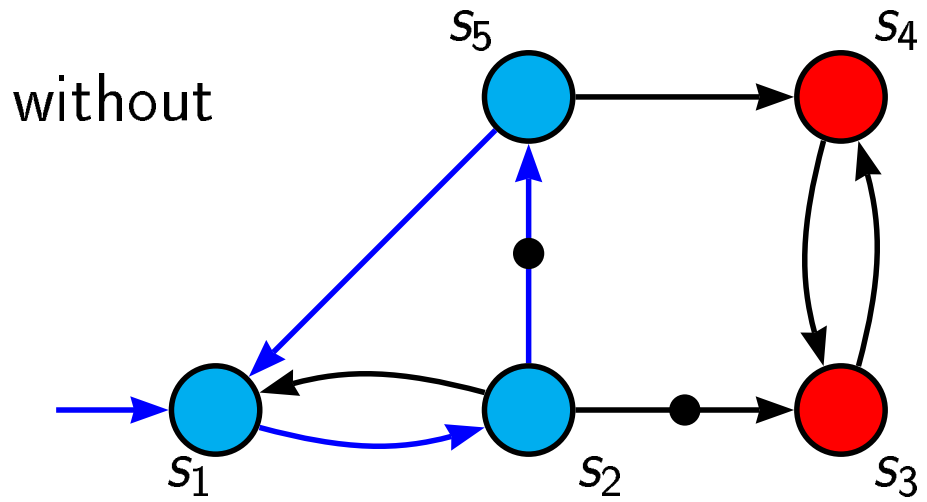
Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



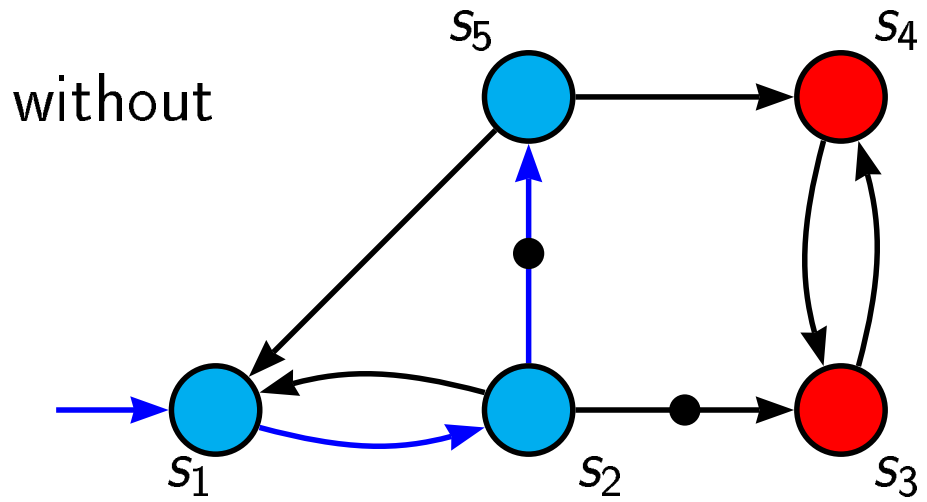
Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



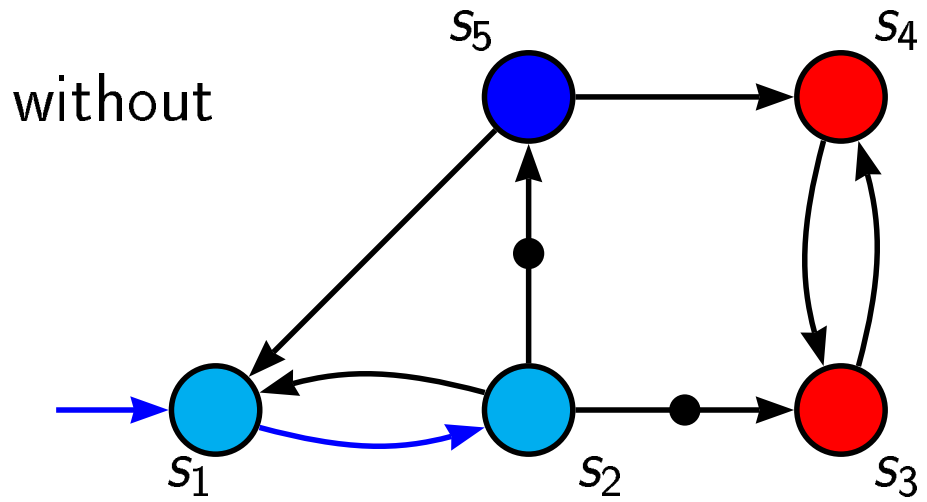
Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



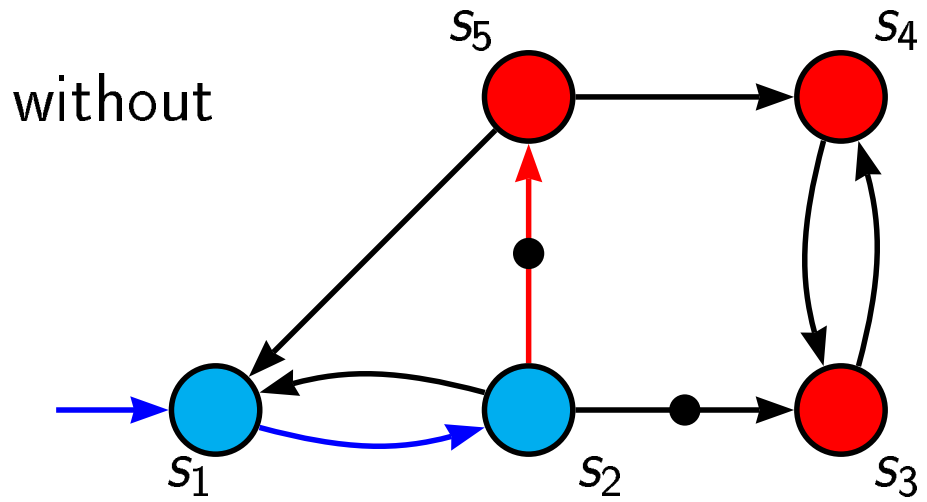
Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



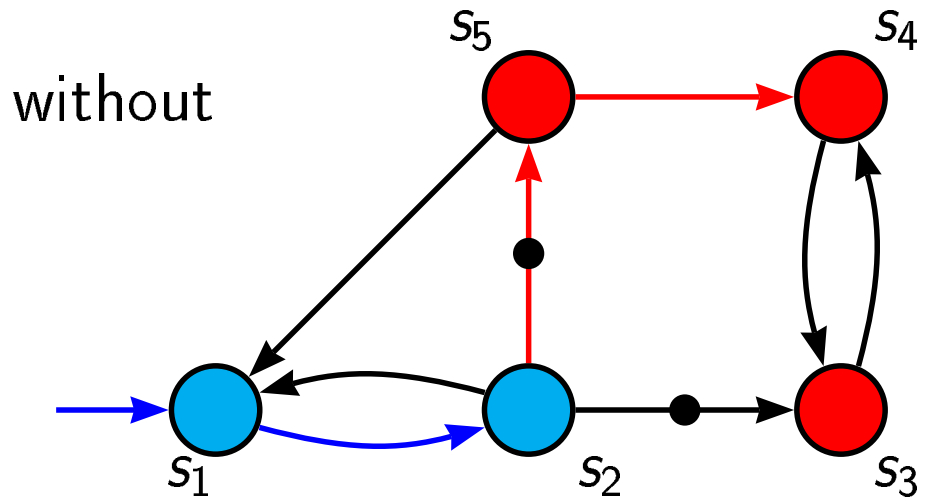
Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



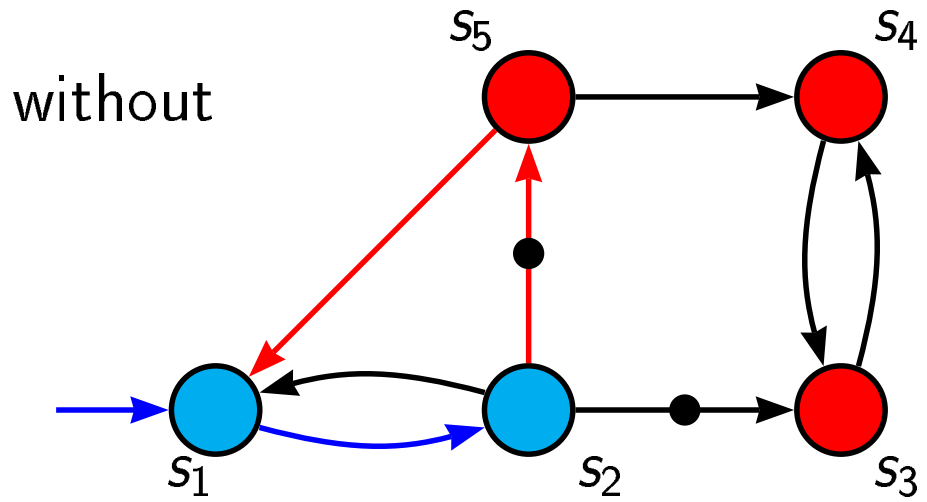
Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



Our Generalized NDFS

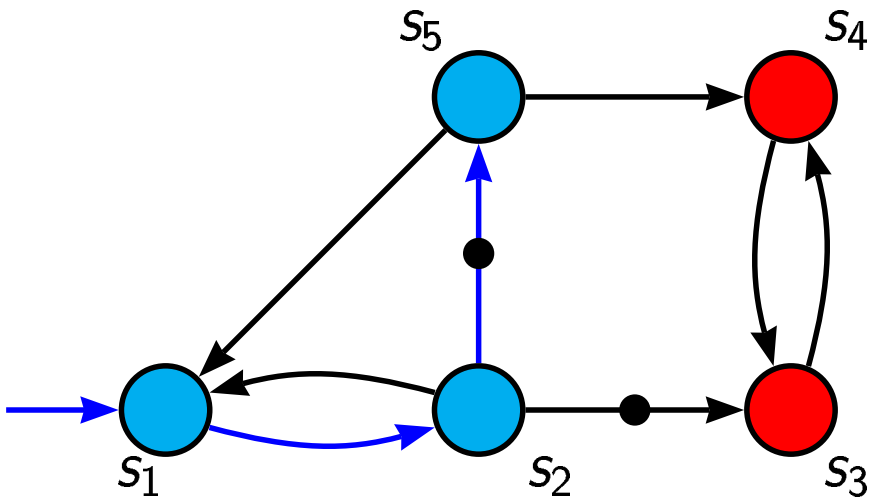
- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



Found!

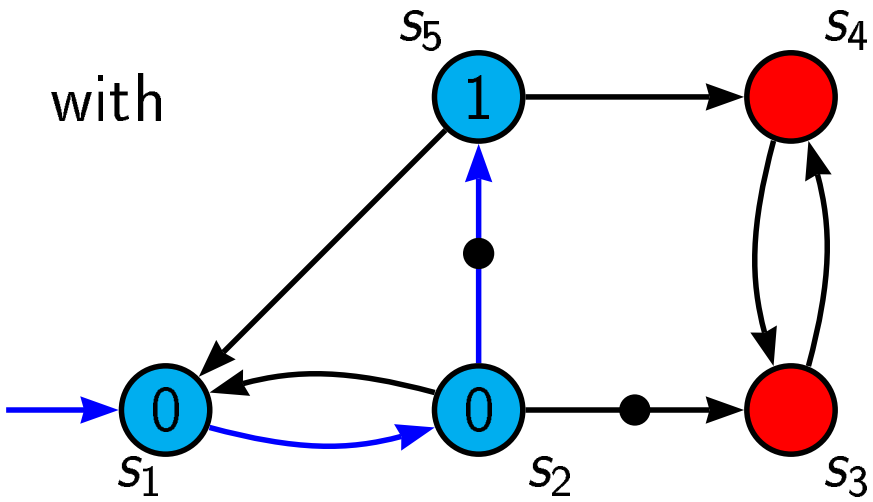
Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



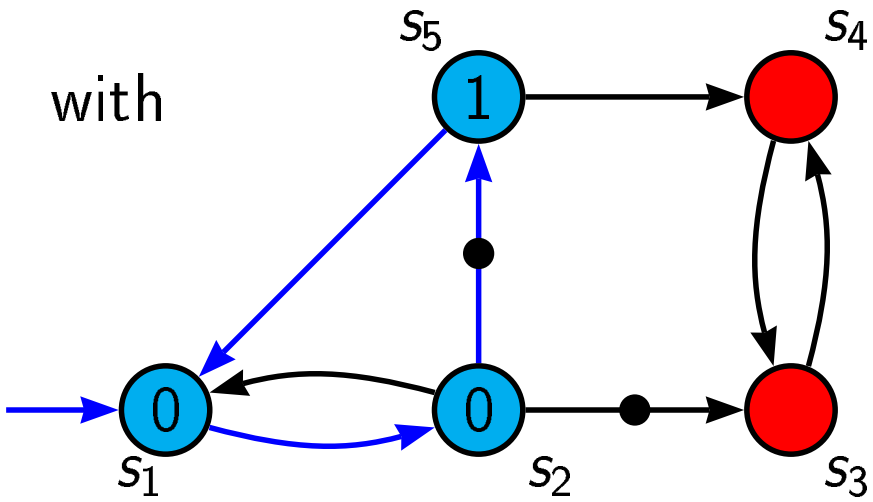
Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



Our Generalized NDFS

- Merge more recent optimizations of Gastin et al. ('04) and Schwoon & Esparza ('04) into Taurainen's algorithm.
- Introduce another optimization: weighted blue stack.



Found!

Generalizable with m counters per state in the blue search.

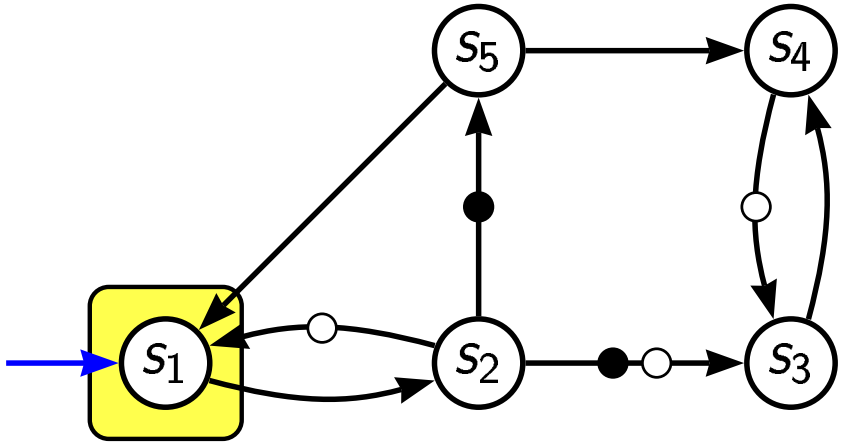
Emptiness Checks History

	degeneralized		generalized	
nested DFS	Courcoubetis et al.	'90		
	Godefroid & Holzmann	'93		
	Holzmann et al.	'96	Tauriainen	'03
	Gastin et al.	'04		
	Schwoon & Esparza	'04	Couvreur et al.	'05

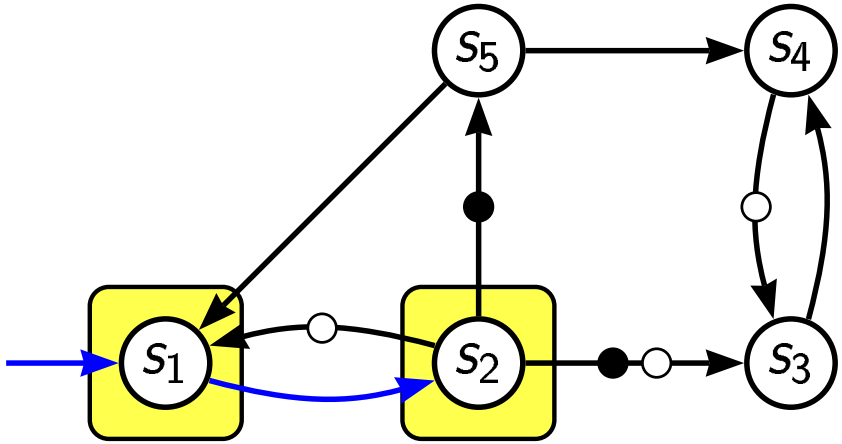
Emptiness Checks History

	degeneralized	generalized
nested DFS	Courcoubetis et al. '90	
	Godefroid & Holzmann '93	
	Holzmann et al. '96	Tauriainen '03
	Gastin et al. '04	
	Schwoon & Esparza '04	Couvreur et al. '05
SCC	Lichtenstein & Pnueli '83	Couvreur '99

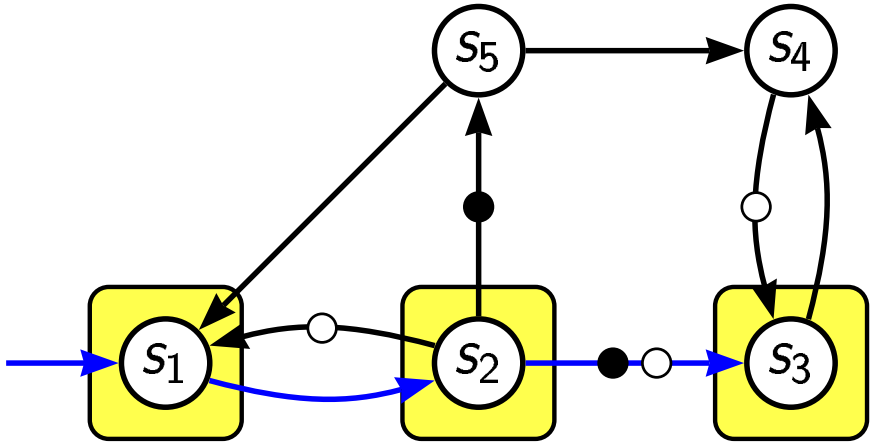
SCC-Based Emptiness Check



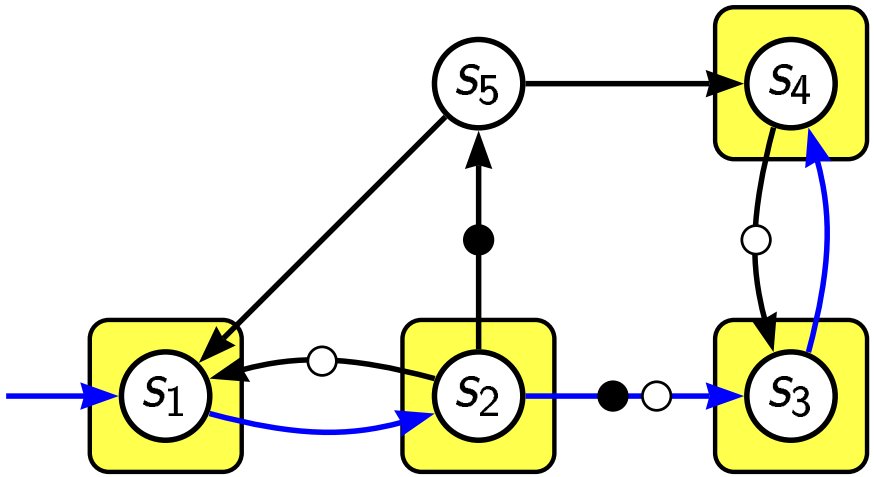
SCC-Based Emptiness Check



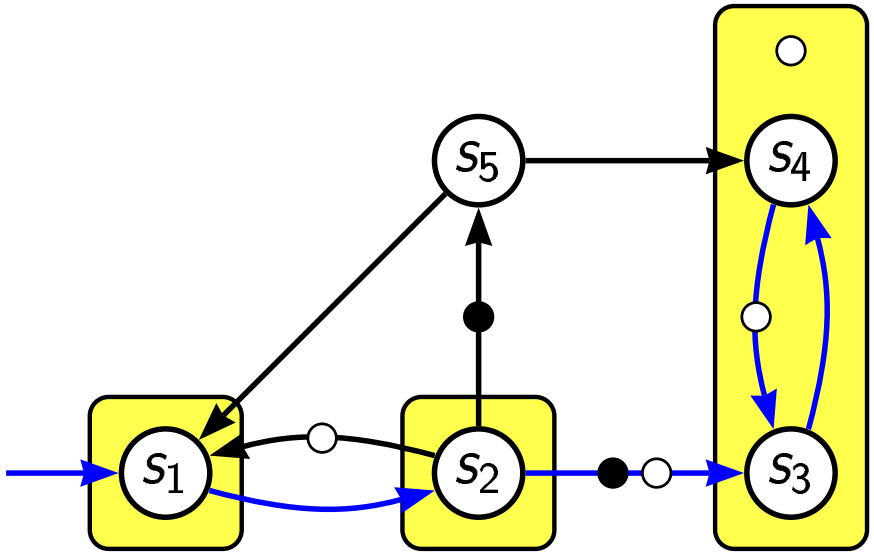
SCC-Based Emptiness Check



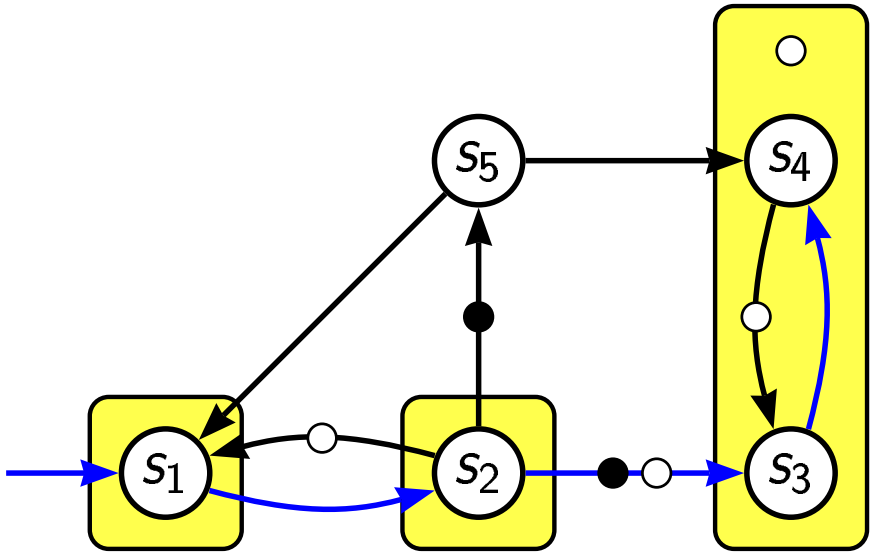
SCC-Based Emptiness Check



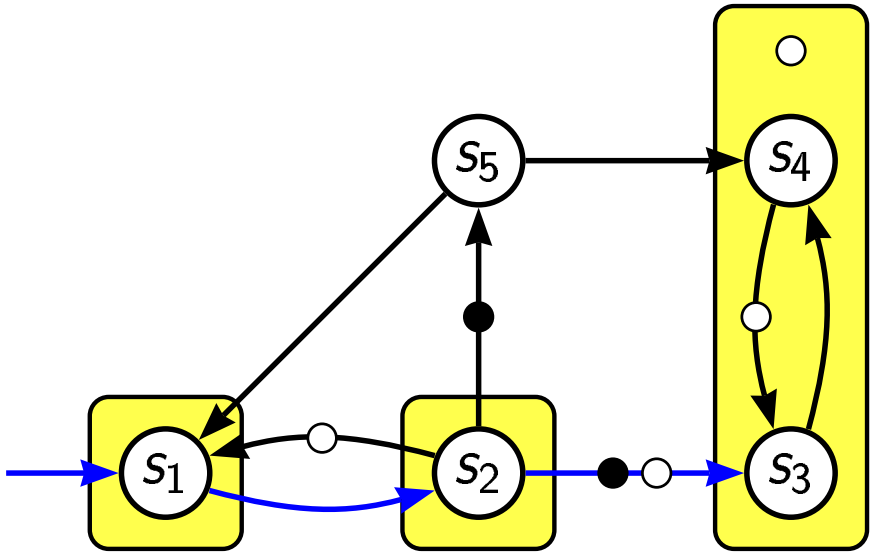
SCC-Based Emptiness Check



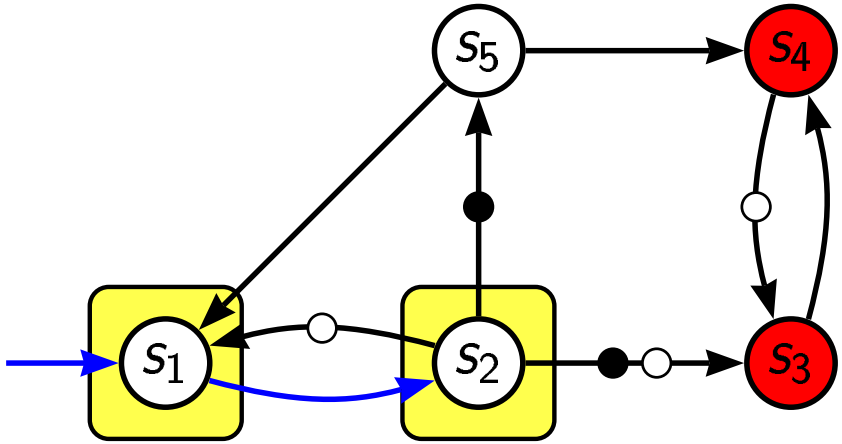
SCC-Based Emptiness Check



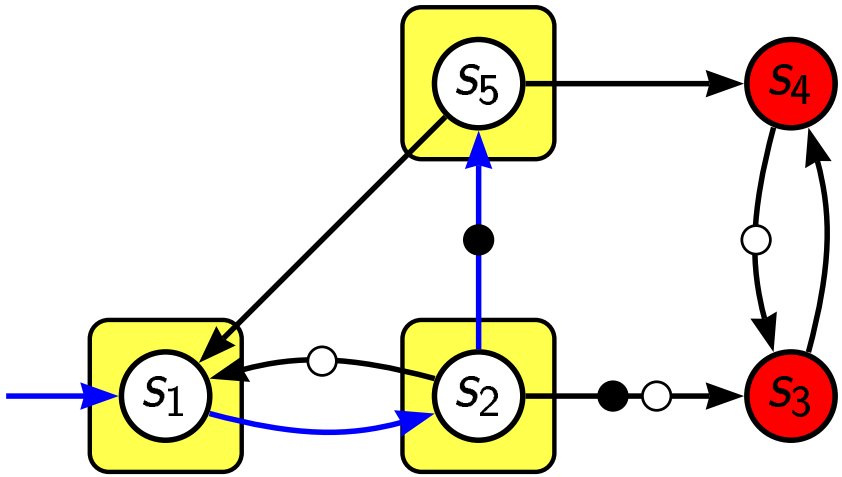
SCC-Based Emptiness Check



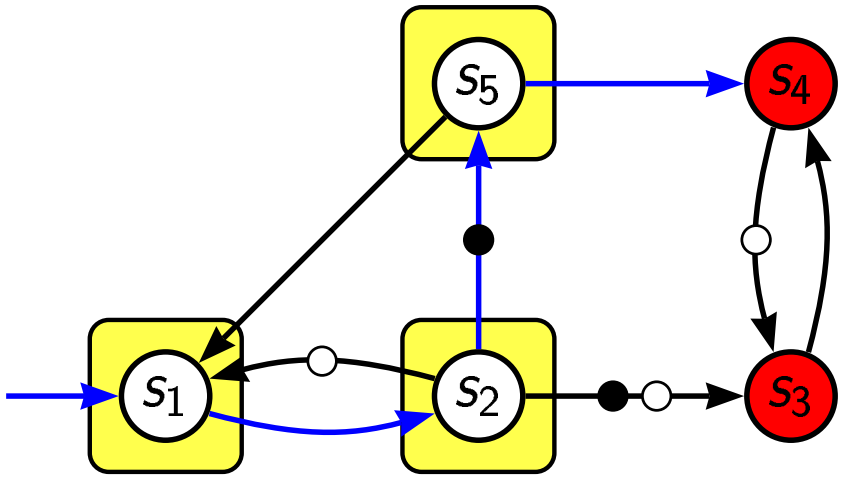
SCC-Based Emptiness Check



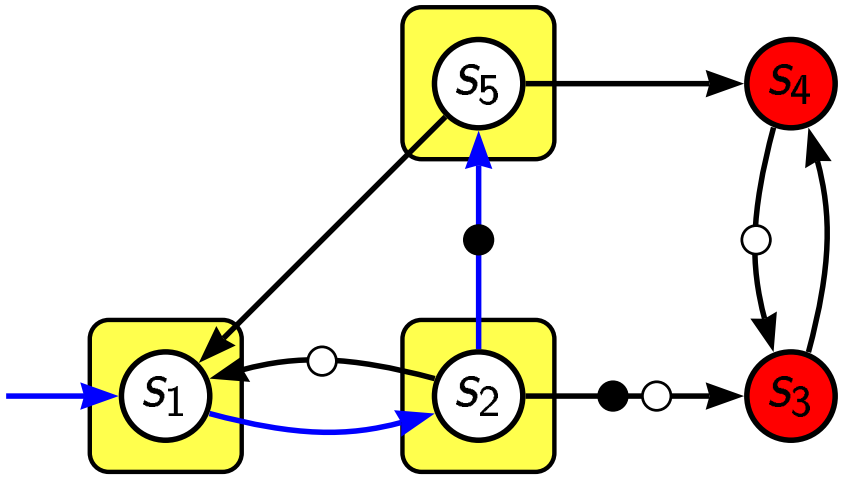
SCC-Based Emptiness Check



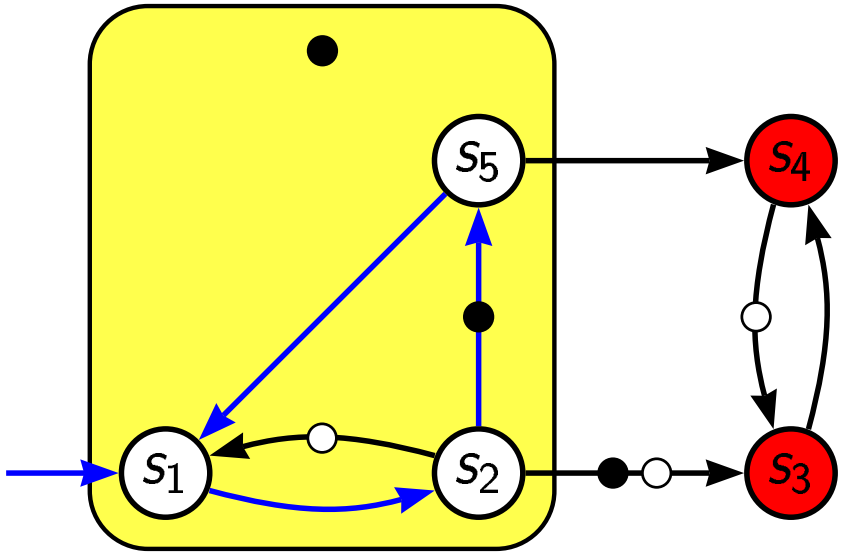
SCC-Based Emptiness Check



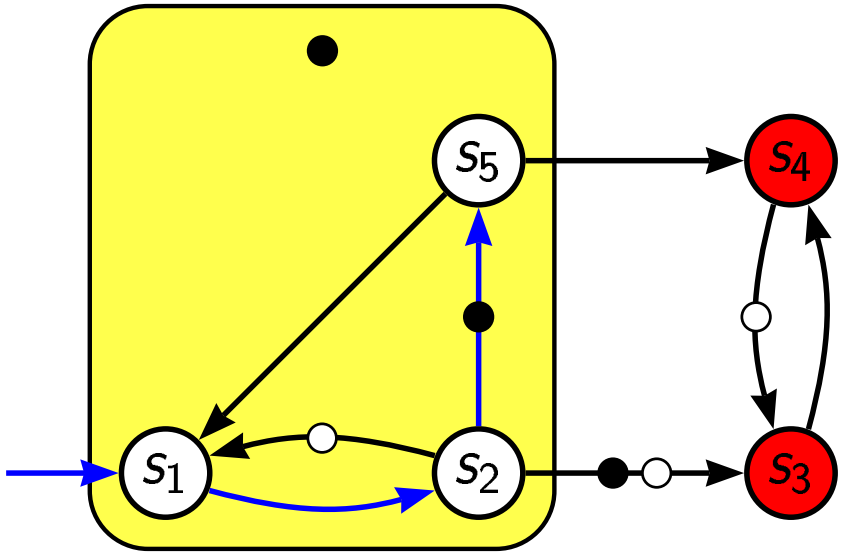
SCC-Based Emptiness Check



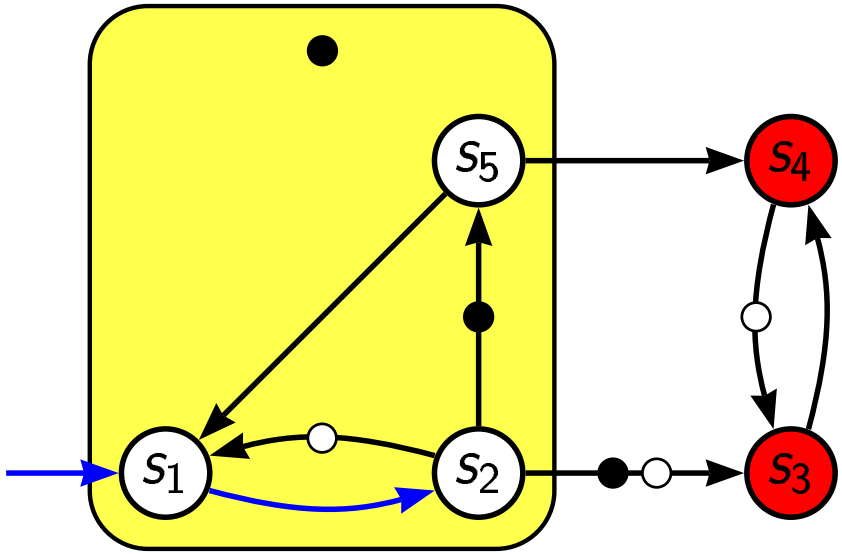
SCC-Based Emptiness Check



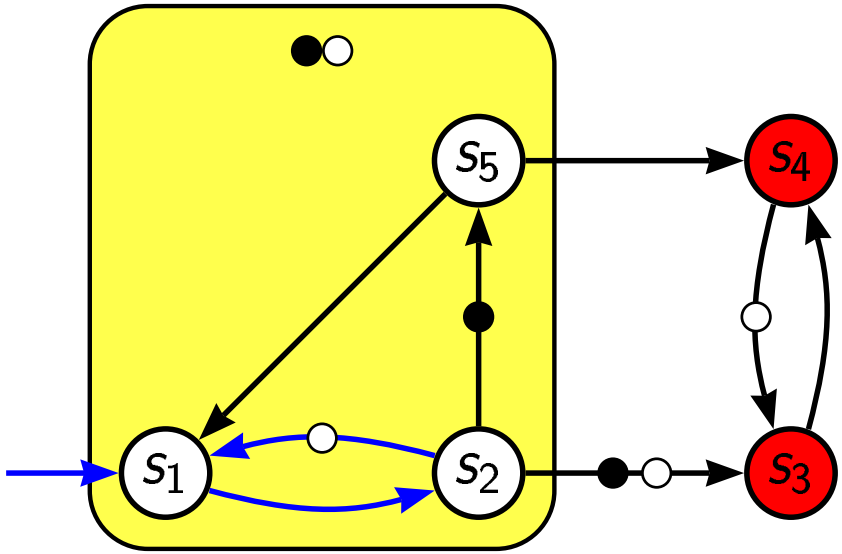
SCC-Based Emptiness Check



SCC-Based Emptiness Check

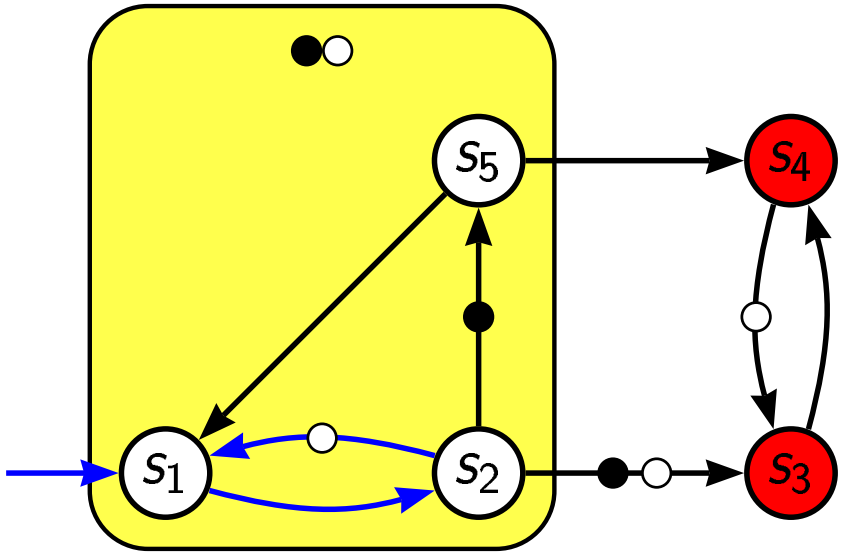


SCC-Based Emptiness Check



Found!

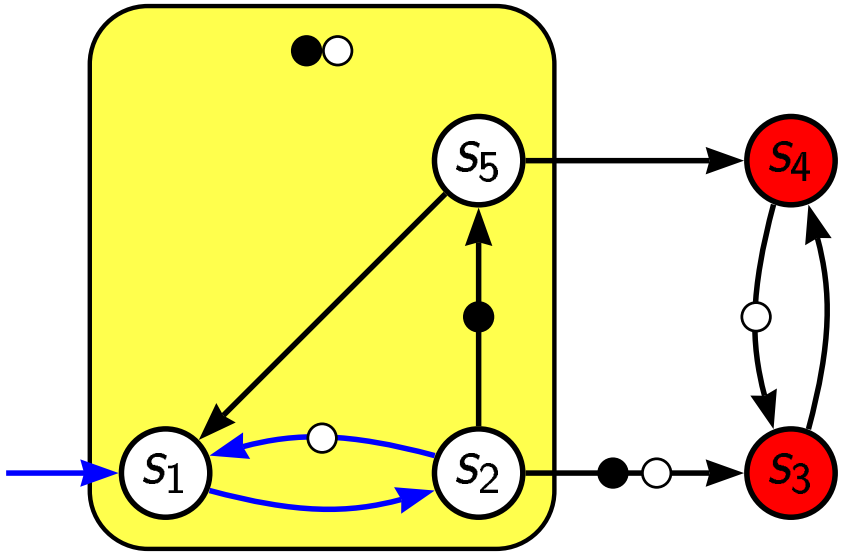
SCC-Based Emptiness Check



Found!

entries in hash table	hash table size in bits	search stack depth	states traversed
n	$n(s_g + \lg n)$	n	$2n$

SCC-Based Emptiness Check



Found!

entries in hash table	hash table size in bits	search stack depth	states traversed
n	$n(s_g + \lg n)$	n	$2n$

Can be reduced to n if the search stack explicitly stores the states of each component (requires more memory).

Emptiness Checks History

	degeneralized		generalized	
nested DFS	Courcoubetis et al.	'90		
	Godefroid & Holzmann	'93		
	Holzmann et al.	'96	Tauriainen	'03
	Gastin et al.	'04		
	Schwoon & Esparza	'04	Couvreur et al.	'05
SCC	Lichtenstein & Pnueli	'83	Couvreur	'99

Emptiness Checks History

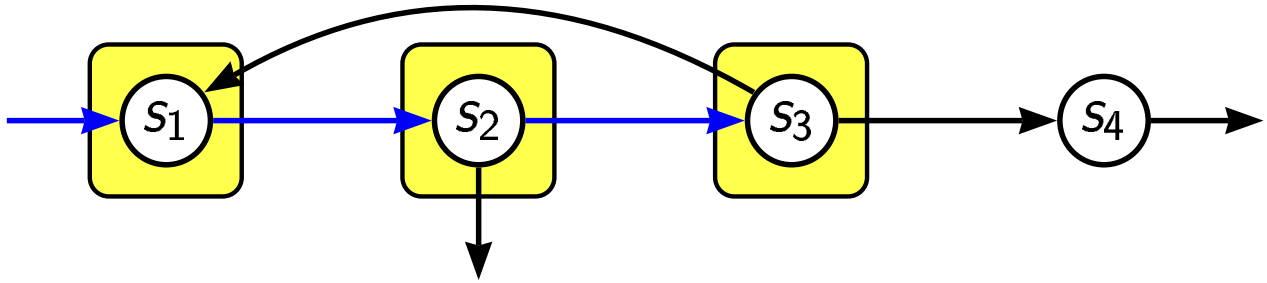
	degeneralized		generalized	
nested DFS	Courcoubetis et al.	'90		
	Godefroid & Holzmann	'93		
	Holzmann et al.	'96	Tauriainen	'03
	Gastin et al.	'04		
	Schwoon & Esparza	'04	Couvreur et al.	'05
SCC	Lichtenstein & Pnueli	'83	Couvreur	'99
	Geldenhuys & Valmari	'04	Geldenhuys & Valmari	'05

Emptiness Checks History

	degeneralized		generalized	
nested DFS	Courcoubetis et al.	'90		
	Godefroid & Holzmann	'93		
	Holzmann et al.	'96	Tauriainen	'03
	Gastin et al.	'04		
	Schwoon & Esparza	'04	Couvreur et al.	'05
SCC	Lichtenstein & Pnueli	'83	Couvreur	'99
	Geldenhuys & Valmari	'04	Geldenhuys & Valmari	'05
			Couvreur et al.	'05

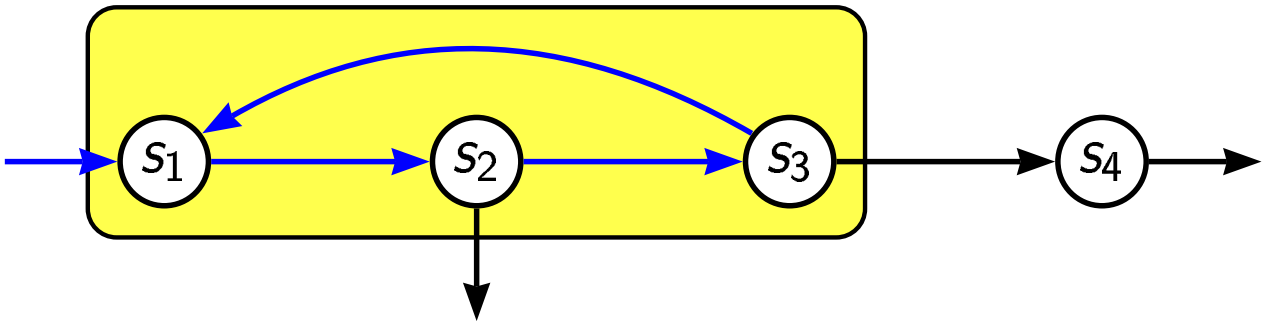
Two Heuristics for SCCs

- H1: visit transitions that go to visited states first.



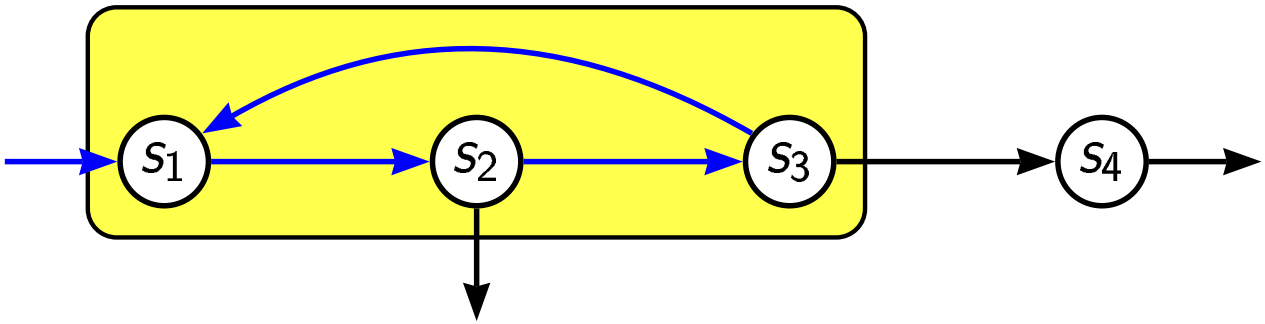
Two Heuristics for SCCs

- H1: visit transitions that go to visited states first.



Two Heuristics for SCCs

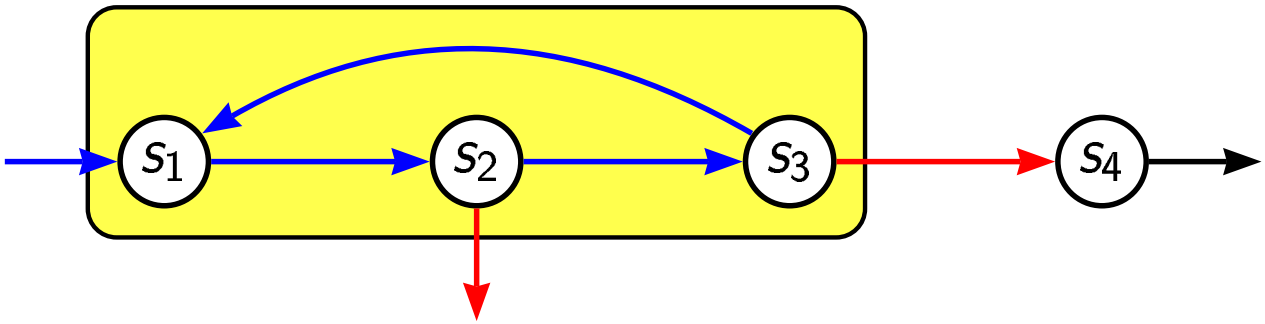
- H1: visit transitions that go to visited states first.



- H2: H1 + consider the DFS in term of SCC when choosing a successor.

Two Heuristics for SCCs

- H1: visit transitions that go to visited states first.



- H2: H1 + consider the DFS in term of SCC when choosing a successor.

Benchmarks

- Upper bounds easy to have.
- Objective : evaluate all these algorithms on the average, on non-empty automata.

	degeneralized		generalized	
nested DFS	Courcoubetis et al.	'90		
	Godefroid & Holzmann	'93		
	Holzmann et al.	'96	Tauriainen	'03
	Gastin et al.	'04		
	Schwoon & Esparza	'04	Couvreur et al.	'05
SCC	Lichtenstein & Pnueli	'83	Couvreur	'99
	Geldenhuys & Valmari	'04	Geldenhuys & Valmari	'05
			Couvreur et al.	'05

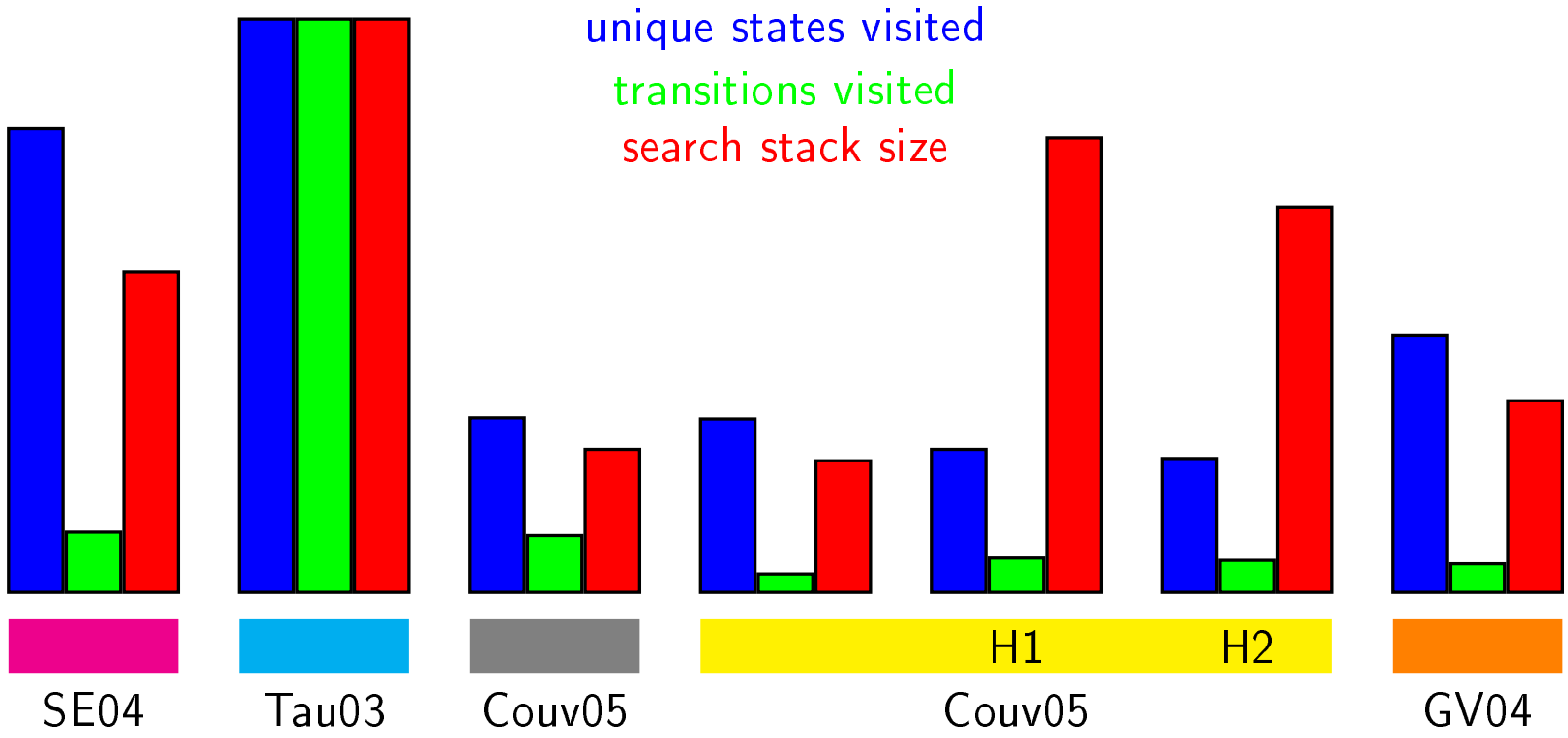
Benchmarks

- Upper bounds easy to have.
- Objective : evaluate all these algorithms on the average, on non-empty automata.

	degeneralized	generalized
nested DFS	Courcoubetis et al. '90	
	Godefroid & Holzmann '93	
	Holzmann et al. '96	Tauriainen '03
	Gastin et al. '04	
	Schwoon & Esparza '04 →	Couvreur et al. '05
SCC	Lichtenstein & Pnueli '83	Couvreur '99
	Geldenhuys & Valmari '04	Geldenhuys & Valmari '05
		Couvreur et al. '05

Benchmarks

nested DFS	Schwoon & Esparza	'04	Tauriainen	'03
			Couvreur et al.	'05
SCC	Geldenhuys & Valmari	'04	Couvreur et al.	'05



Conclusions

- generalized vs. non-generalized:
  - generalized algorithms require less memory
  - generalized algorithms produce more meaningful counterexamples
  - weak fairness expressible using generalized conditions
  - non-generalized NDFSs produce counterexamples directly
- NDFS vs. SCC algorithms:
  - SCC algorithms check emptiness faster
  - SCC algorithms scale to generalized conditions and fairness conditions easily
  - NDFSs require less memory
- All these algorithms and the benchmark framework are implemented in our model checking library:

  `http://spot.lip6.fr`

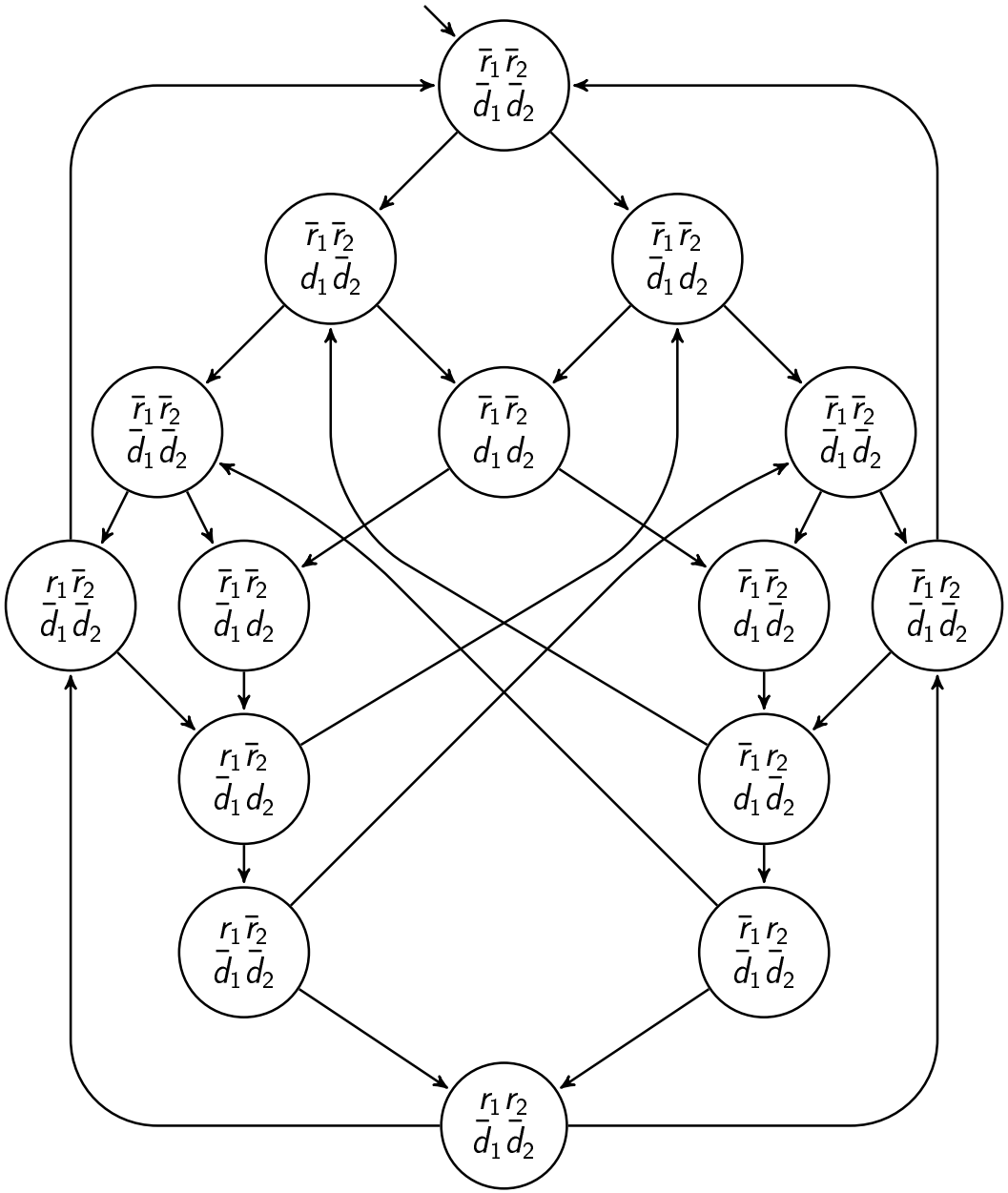
Part II

Partial Order Methods

Mostly based on Section 4 of:

Marko Rauhamaa
A comparative study of methods for efficient reachability analysis.
Helsinki University of Technology, Research Report A14, September 1990.
`www.tcs.hut.fi/Publications/bibdb/HUT-TCS-A14.ps`

Kripke Structure to Simplify



Can't we simplify?

We want to verify $\mathbf{G}(d_1 \rightarrow \mathbf{F}\, r_1)$.

On this formula, the following two executions are equivalent:

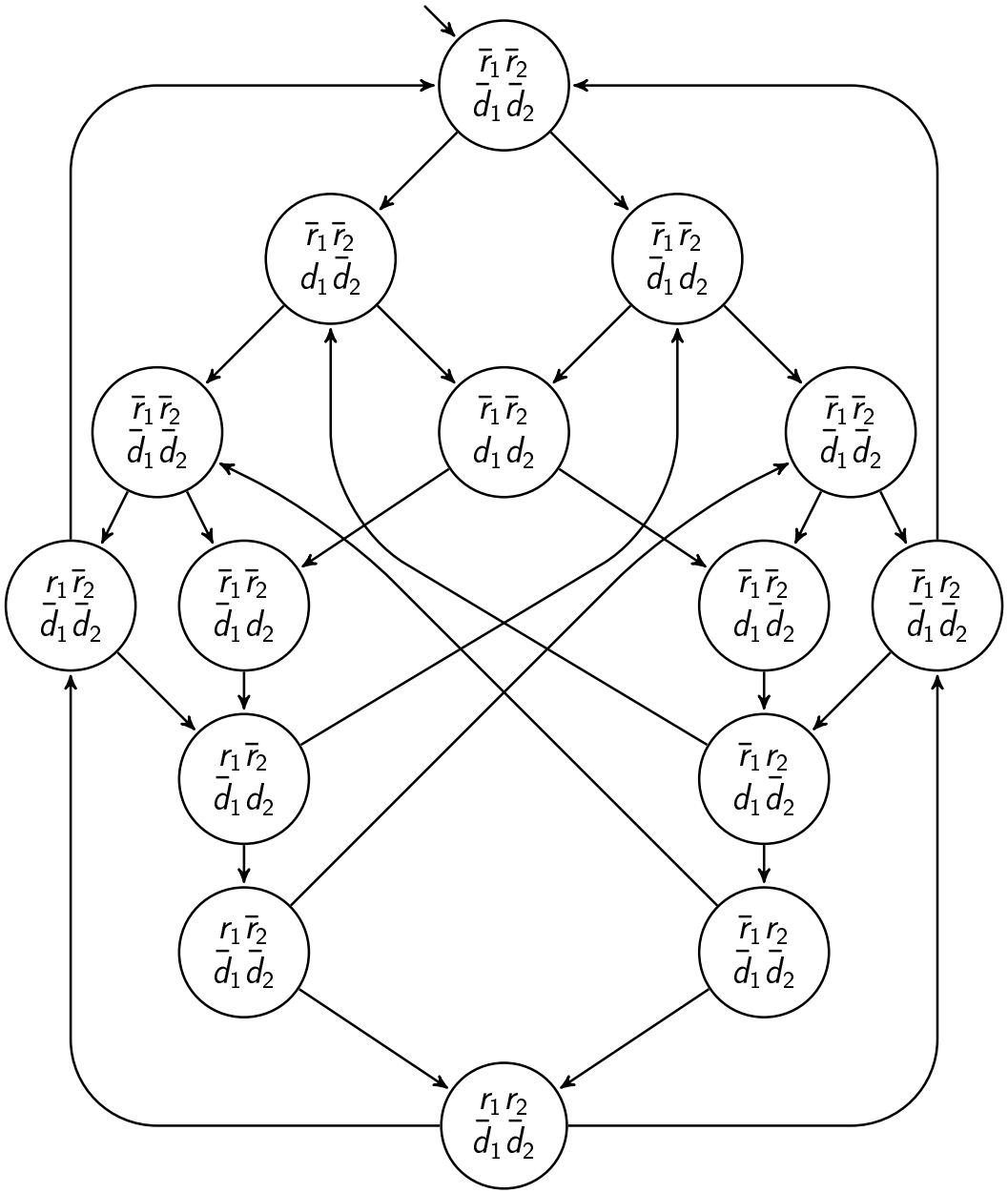
- client C_1 sends a request
- client C_2 sends a request
- other events...

- client C_2 sends a request
- client C_1 sends a request
- other events (in same order)

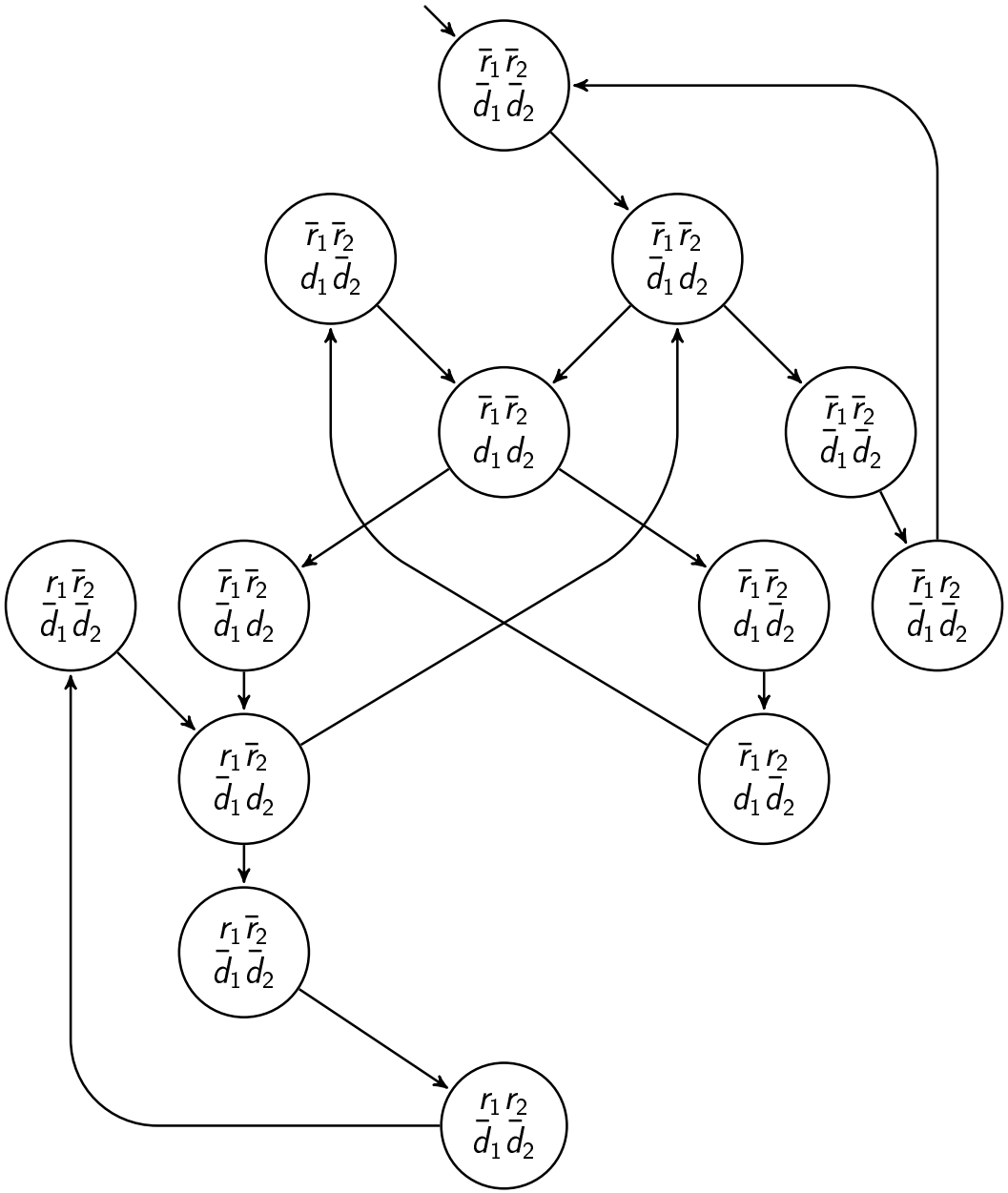
The order between the two requests does not make any difference.
Some notes:

- C_1's request has an influence on d_1 (*observed* by the formula),
- C_2's request has no influence on the formula (*unobserved event*),
- the two events are *independent* (doing one will not prevent the other)

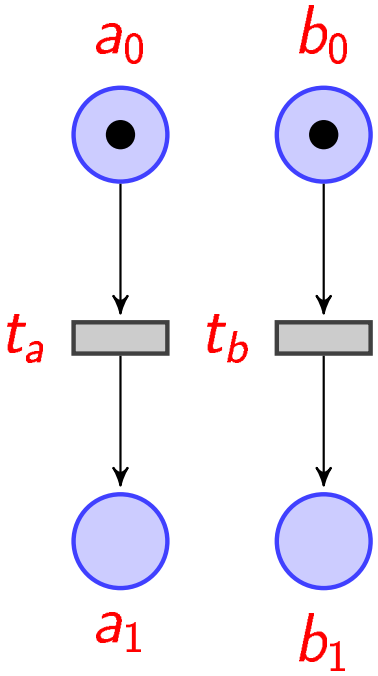
Kripke Structure to Simplify

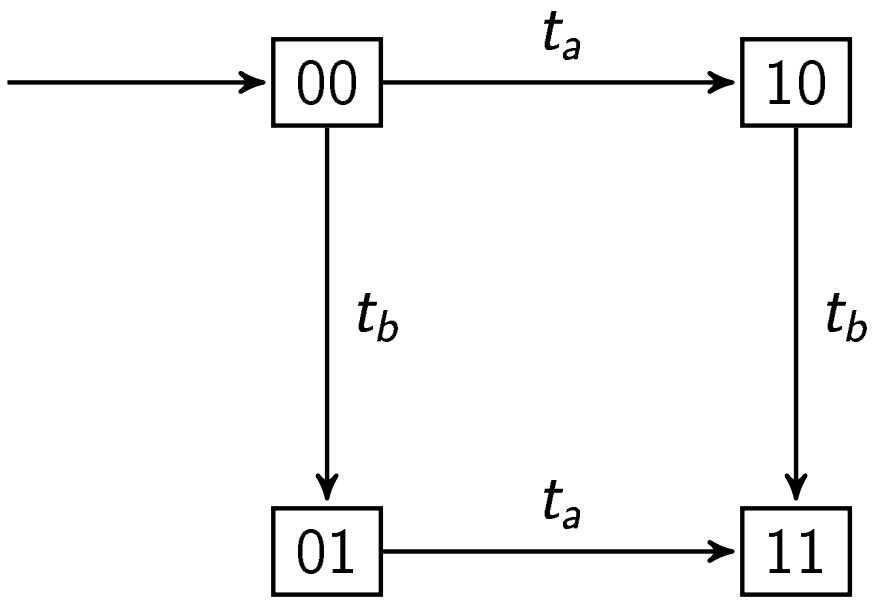


Kripke Structure to Simplify

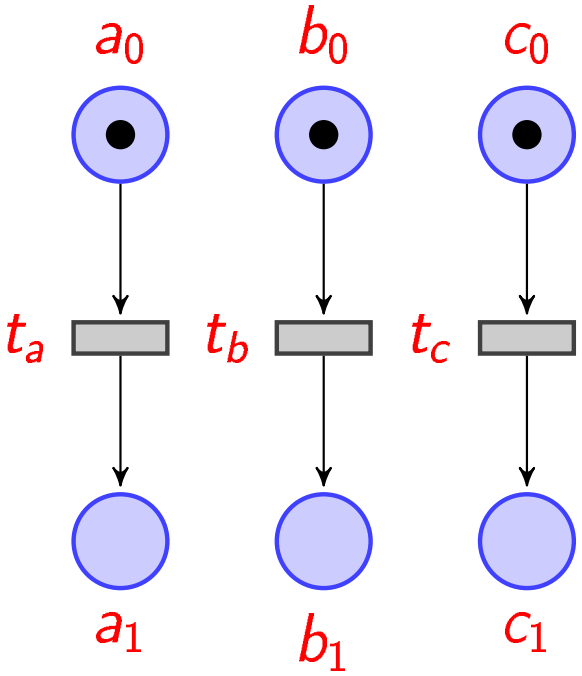


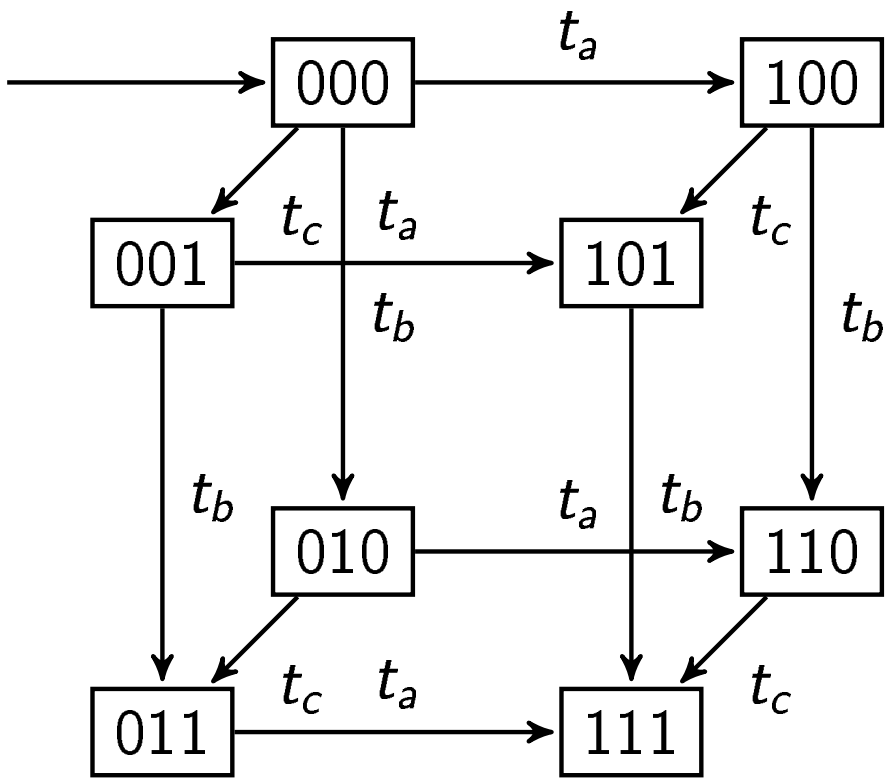
Let's start (very) simple



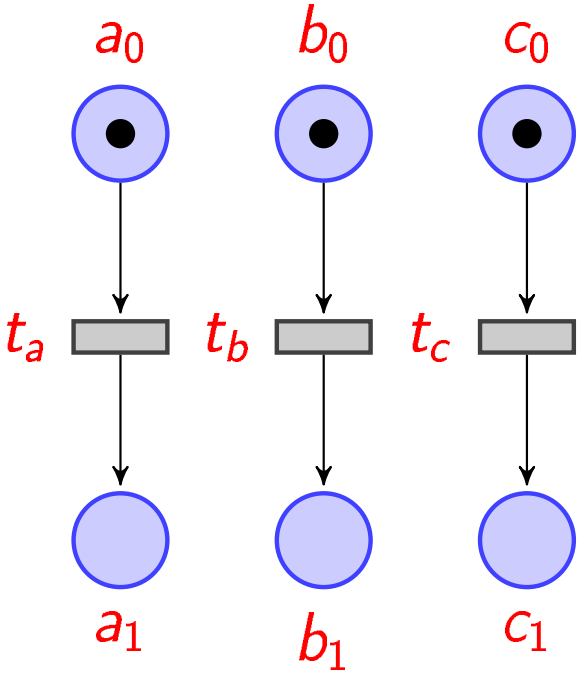


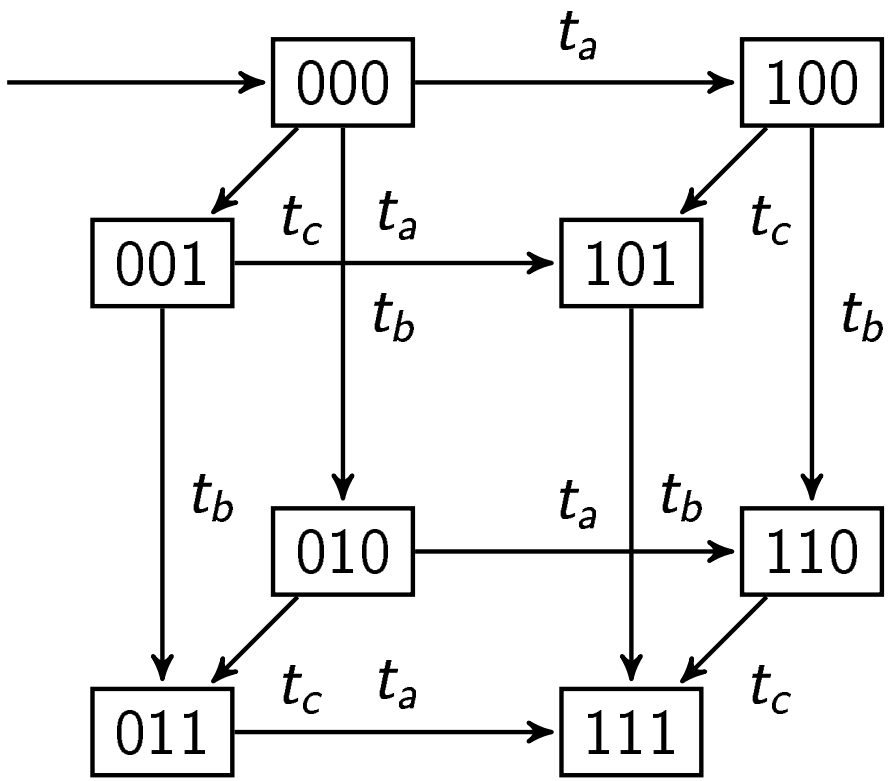
Let's start (very) simple





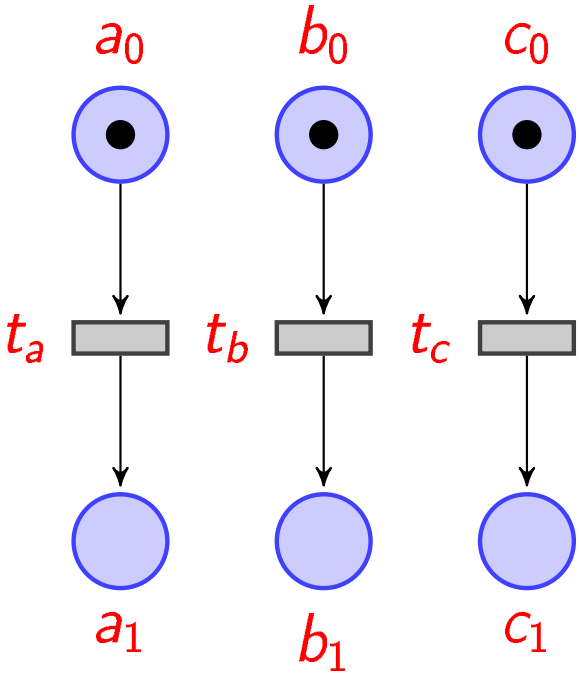
Let's start (very) simple

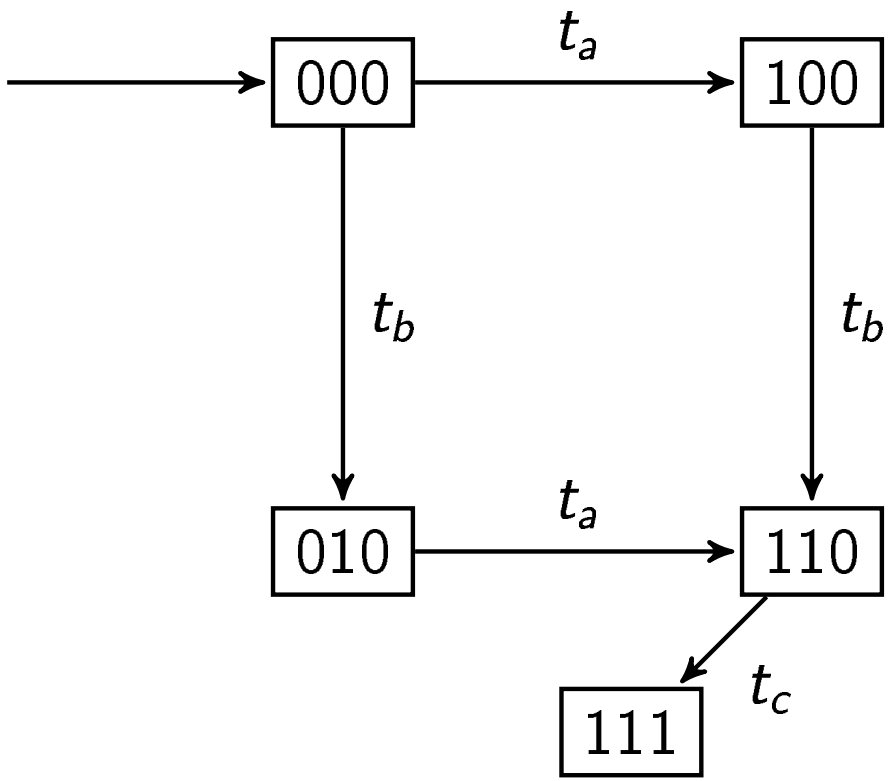




For any sequence $M \xrightarrow{\sigma t_c \sigma'} M'$, there exists a sequence $M \xrightarrow{\sigma \sigma' t_c} M'$.
If we do not observe t_c, we can keep only the latter.

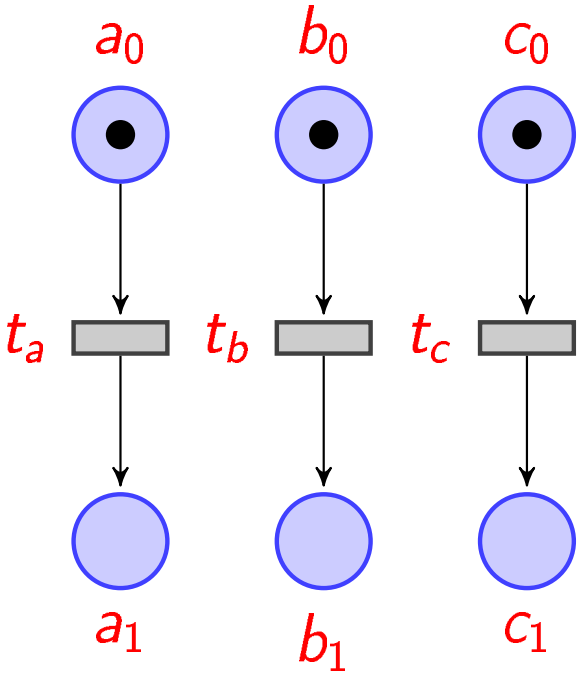
Let's start (very) simple

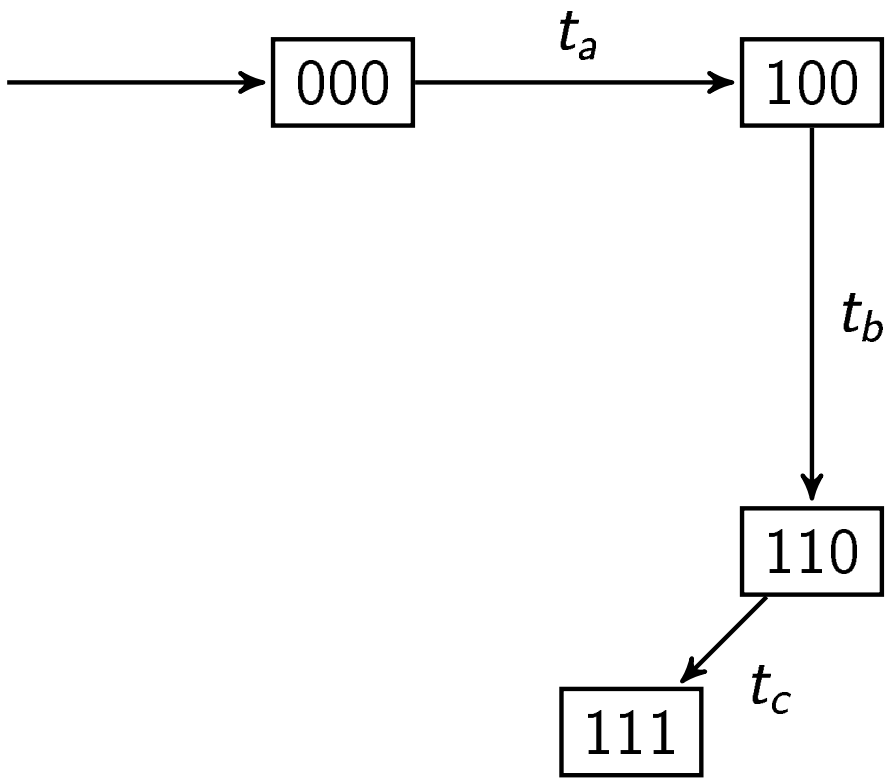




For any sequence $M \xrightarrow{\sigma t_c \sigma'} M'$, there exists a sequence $M \xrightarrow{\sigma \sigma' t_c} M'$.
If we do not observe t_c, we can keep only the latter.

Let's start (very) simple





For any sequence $M \xrightarrow{\sigma t_c \sigma'} M'$, there exists a sequence $M \xrightarrow{\sigma \sigma' t_c} M'$. If we do not observe t_c, we can keep only the latter. Of course if we do not observe t_b we can further simplify the graph.

The Priority Method

- A way to reduce state space with *a priori* knowledge.
- The analyst manually supplies a *partial ordering* of the events.
  E.g. $t_a \prec t_b \prec t_c$.
- This ordering is used whenever there is a choice between events.
- The information is *static*.
- What kind of property does it preserves?
  (assuming the partial ordering has been set properly)

The Priority Method

- A way to reduce state space with *a priori* knowledge.
- The analyst manually supplies a *partial ordering* of the events.
  E.g. $t_a \prec t_b \prec t_c$.
- This ordering is used whenever there is a choice between events.
- The information is *static*.
- What kind of property does it preserves?
  (assuming the partial ordering has been set properly)
  - Reachability of a state?

The Priority Method

- A way to reduce state space with *a priori* knowledge.
- The analyst manually supplies a *partial ordering* of the events.
  E.g. $t_a \prec t_b \prec t_c$.
- This ordering is used whenever there is a choice between events.
- The information is *static*.
- What kind of property does it preserves?
  (assuming the partial ordering has been set properly)
  - Reachability of a state? Not all, obviously

The Priority Method

- A way to reduce state space with *a priori* knowledge.
- The analyst manually supplies a *partial ordering* of the events.
  E.g. $t_a \prec t_b \prec t_c$.
- This ordering is used whenever there is a choice between events.
- The information is *static*.
- What kind of property does it preserves?
  (assuming the partial ordering has been set properly)
  - Reachability of a state? Not all, obviously
  - Deadlock detection?

The Priority Method

- A way to reduce state space with *a priori* knowledge.
- The analyst manually supplies a *partial ordering* of the events.
  E.g. $t_a \prec t_b \prec t_c$.
- This ordering is used whenever there is a choice between events.
- The information is *static*.
- What kind of property does it preserves?
  (assuming the partial ordering has been set properly)
  - Reachability of a state? Not all, obviously
  - Deadlock detection? Yes

The Priority Method

- A way to reduce state space with *a priori* knowledge.
- The analyst manually supplies a *partial ordering* of the events.
  E.g. $t_a \prec t_b \prec t_c$.
- This ordering is used whenever there is a choice between events.
- The information is *static*.
- What kind of property does it preserves?
  (assuming the partial ordering has been set properly)
  - Reachability of a state? Not all, obviously
  - Deadlock detection? Yes
  - Verification of LTL formulæ?
  - Verification of LTL$\setminus \mathbf{X}$ formulæ?

The Priority Method

- A way to reduce state space with *a priori* knowledge.
- The analyst manually supplies a *partial ordering* of the events.
  E.g. $t_a \prec t_b \prec t_c$.
- This ordering is used whenever there is a choice between events.
- The information is *static*.
- What kind of property does it preserves?
  (assuming the partial ordering has been set properly)
  - Reachability of a state? Not all, obviously
  - Deadlock detection? Yes
  - Verification of LTL formulæ? Some
  - Verification of LTL$\setminus \mathbf{X}$ formulæ? Some

We need to formalize the concepts of *dependent* and *observed* events, and introduce *dynamic ordering*.

A Definition for Petri Nets

$\langle S, T, W, M_0 \rangle$ where

- S is the set of states,
- T is the set of transitions,
- $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ is the arc weight function,
- $M_0 : S \to \mathbb{N}$ is the initial marking.

For $x \in S \times T$ we denote $\bullet x = \{y \mid W(y, x) > 0\}$.
For $x \in T \times S$ we denote $x\bullet = \{y \mid W(x, y) > 0\}$.
A marking is a $S \to \mathbb{N}$ function.
A transition $t \in T$ is enabled at marking M (denoted $M \xrightarrow{t}$) if $\forall s \in \bullet t,\ M(s) \geq W(s, t)$.
A transition t enabled at marking M can fire into marking M' (denoted $M \xrightarrow{t} M'$) if $\forall s \in S,\ M'(s) = M(s) - W(s, t) + W(t, s)$.
A marking M is dead if $\nexists t \in T,\ M \xrightarrow{t}$.

The Reachability Graph

The reachability graph of a Petri net $\langle S, T, W, M_0 \rangle$ is a pair $\langle V, E \rangle$ where

- V (vertices) is a set of markings,
- $E \subset V \times T \times V$ (edges)

and the following hold

- $M_0 \in V$,
- if $M \in V$ and $M \xrightarrow{t} M'$ then $M' \in V$ and $(M, t, M') \in E$,
- V and E contain no other elements.

Note that if $M \xrightarrow{tt'} M'$ and $M \xrightarrow{t't}$, then $M \xrightarrow{t't} M'$.

Some Abbreviations

For $\sigma = t_1 t_2 \cdots t_n \in T^\star$ we denote $M \xrightarrow{\sigma}$ is there exists M_1, M_2, $\ldots M_{n-1}$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots M_{n-1} \xrightarrow{t_n}$.

Similarly $M \xrightarrow{\sigma} M'$ if there additionally exists M' such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots M_{n-1} \xrightarrow{t_n} M'$

The Idea behind Stubborn Sets

The Petri net is split in two parts: a black box and an environment, such that the transitions of the two sets are independent.

In the following we assume that t is a black box transition while σ is a sequence of transitions from the environment.

Principle 1 If $\neg M \xrightarrow{T}$ and $M \xrightarrow{\sigma}$, then $\neg M \xrightarrow{\sigma t}$.
In other words, firing transitions in the environment cannot enable a disabled transition of the black box.

Principle 2 If $M \xrightarrow{T}$ and $M \xrightarrow{\sigma}$, then $M \xrightarrow{\sigma t}$ and $M \xrightarrow{t\sigma}$.
Transition from the environment and from the black box can be interleaved as wished.

Application to Finding Dead Markings

P_1 If $\neg M \xrightarrow{T}$ and $M \xrightarrow{\sigma}$, then $\neg M \xrightarrow{\sigma t}$.

P_2 If $M \xrightarrow{T}$ and $M \xrightarrow{\sigma}$, then $M \xrightarrow{\sigma t}$ and $M \xrightarrow{t\sigma}$.

When looking for dead states, we can simplify the reachability graph by firing any black box transitions (t) before environment transitions (σ).

Let there be an enabled transition r in the black box and a path π leading to a dead marking.
Then π must contain some transition t from the black box.
(Otherwise, by P_1, π cannot use non enabled transitions of the black box, and by P_2 the transition r would still be fireable on the dead marking...)
P_2 allows us to move t to the front of π.

Can be move t to the back of π?

Application to Finding Dead Markings

P_1 If $\neg M \xrightarrow{T}$ and $M \xrightarrow{\sigma}$, then $\neg M \xrightarrow{\sigma t}$.

P_2 If $M \xrightarrow{T}$ and $M \xrightarrow{\sigma}$, then $M \xrightarrow{\sigma t}$ and $M \xrightarrow{t\sigma}$.

When looking for dead states, we can simplify the reachability graph by firing any black box transitions (t) before environment transitions (σ).

Let there be an enabled transition r in the black box and a path π leading to a dead marking.
Then π must contain some transition t from the black box.
(Otherwise, by P_1, π cannot use non enabled transitions of the black box, and by P_2 the transition r would still be fireable on the dead marking...)
P_2 allows us to move t to the front of π.

Can be move t to the back of π? Yes.

Looser Principles ⇒ Stubborn Sets

Again t is a transition from the black box (stubborn set T_M), and σ is a sequence of transitions from the environment ($T \setminus T_M$).

Principle 1* If $M \xrightarrow{\sigma t}$, then $M \xrightarrow{t\sigma}$.
Transitions of the stubborn set can be moved before those of the environment.

Principle 2* If $M \xrightarrow{\sigma}$, then $M \xrightarrow{\sigma t'}$ for some fixed transition $t' \in T_M$.
In other words the stubborn set is never empty and the environment cannot disable its transitions.

When looking for dead states, we can still simplify the reachability graph by firing transitions from the stubborn set before any other.

Dead Markings with these Definitions

P_1^* If $M \xrightarrow{\sigma t}$, then $M \xrightarrow{t\sigma}$.

P_2^* If $M \xrightarrow{\sigma}$, then $M \xrightarrow{\sigma t'}$.

Let $M \xrightarrow{\pi} M'$ be a transition sequence to a dead marking M'.
π necessarily contain a transition from the stubborn set. (If it does not, P_2^* implies that $M' \xrightarrow{t'}$ and M' cannot be dead.)
Therefore $\pi = \sigma t \pi'$ and by P_1^* we have $M \xrightarrow{t\sigma\pi'} M'$.

Can be move t to the back of π?

Dead Markings with these Definitions

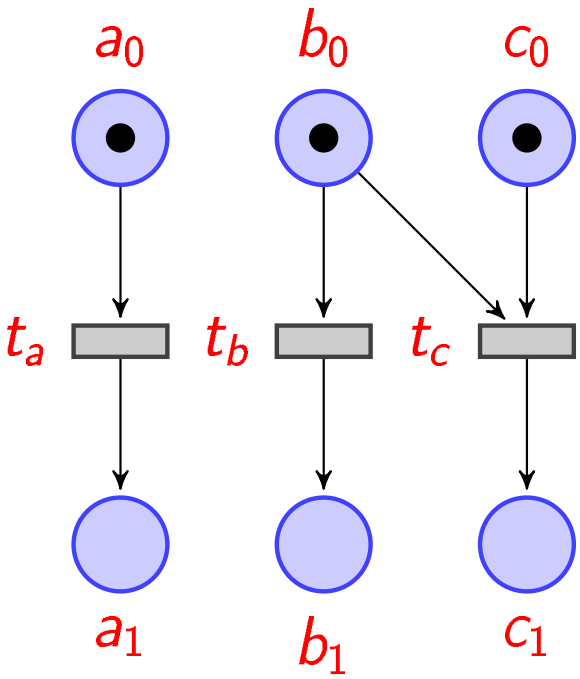
P_1^* If $M \xrightarrow{\sigma t}$, then $M \xrightarrow{t\sigma}$.

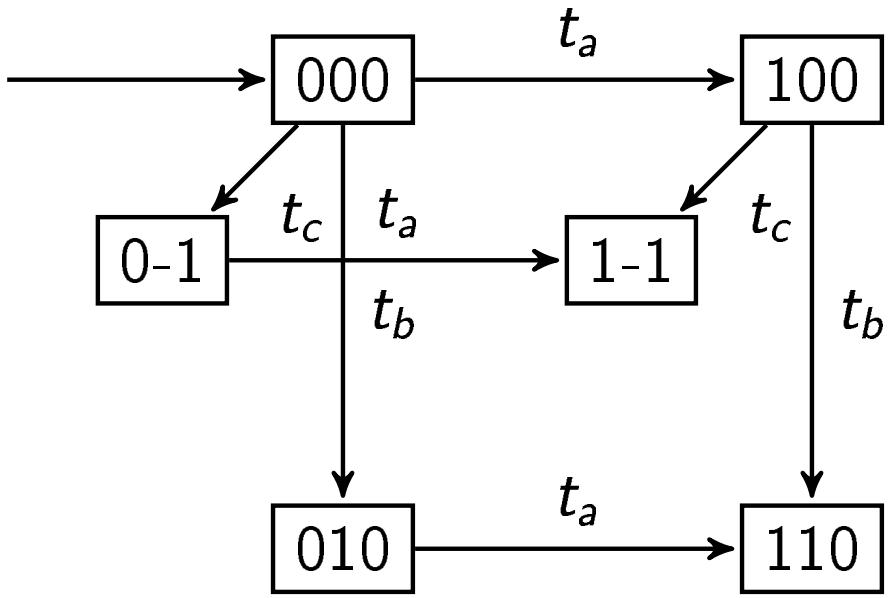
P_2^* If $M \xrightarrow{\sigma}$, then $M \xrightarrow{\sigma t'}$.

Let $M \xrightarrow{\pi} M'$ be a transition sequence to a dead marking M'.
π necessarily contain a transition from the stubborn set. (If it does not, P_2^* implies that $M' \xrightarrow{t'}$ and M' cannot be dead.)
Therefore $\pi = \sigma t \pi'$ and by P_1^* we have $M \xrightarrow{t\sigma\pi'} M'$.

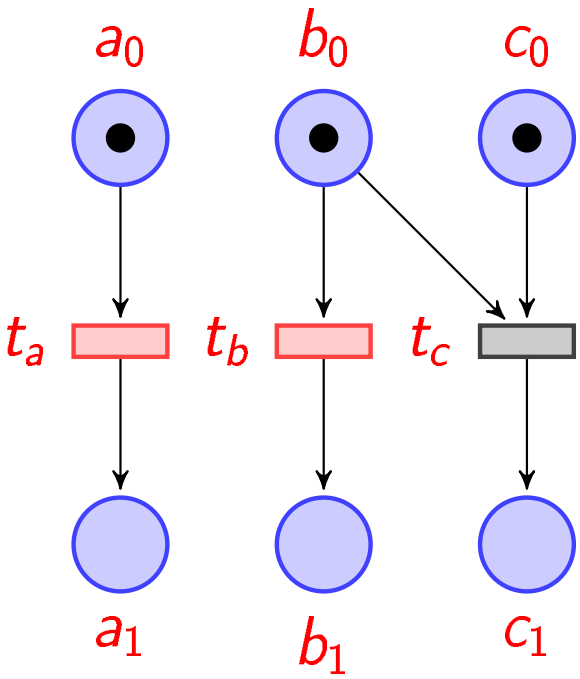
Can be move t to the back of π? No: some transitions of π' are allowed to disable t.

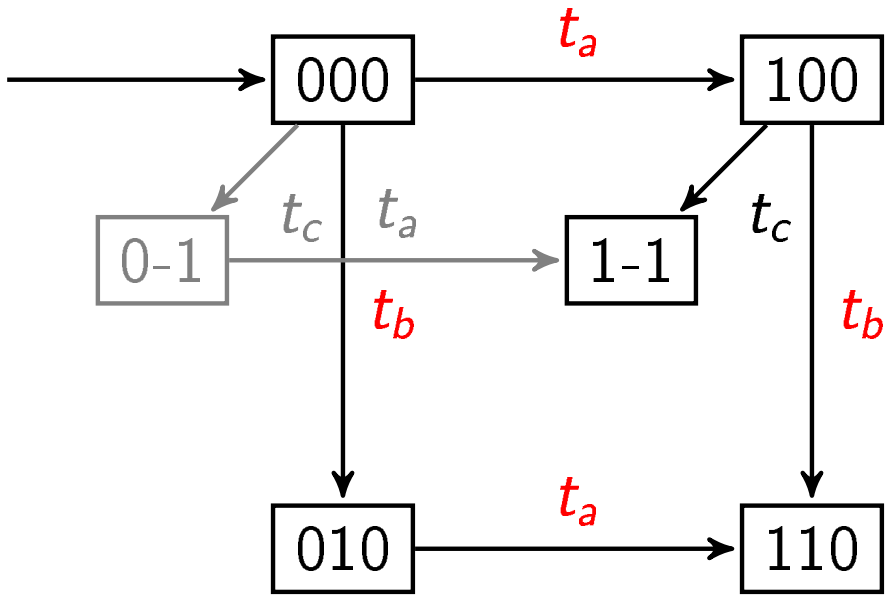
Exemple Stubborn Set





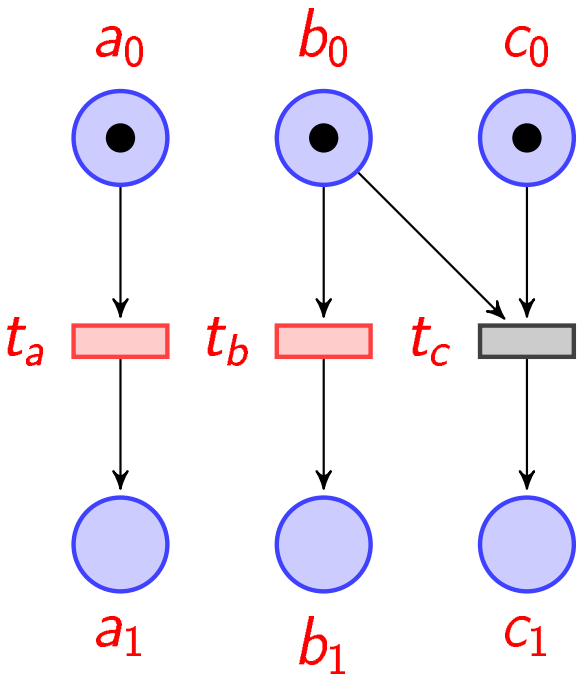
Exemple Stubborn Set

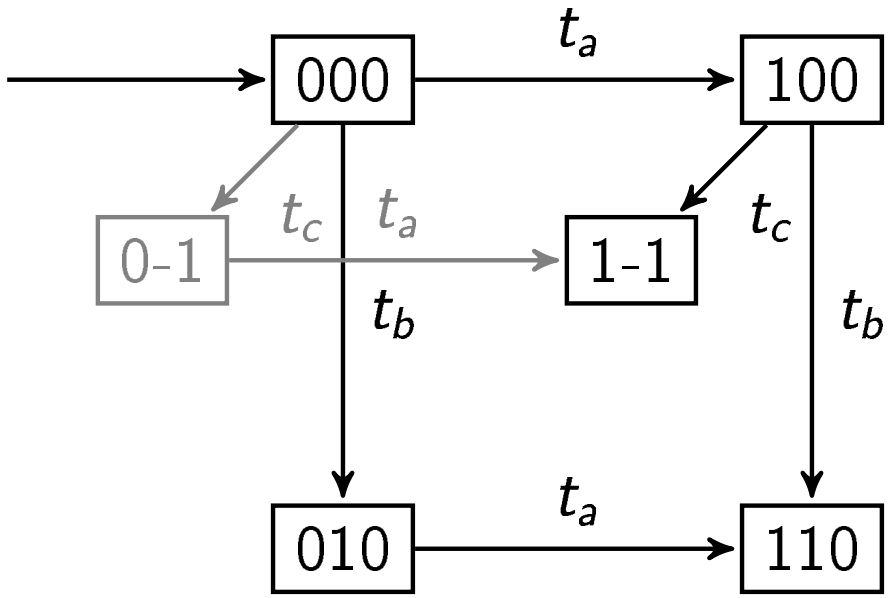




Ex:
Stubborn set for marking 000: $T_{000} = \{t_a, t_b\}$.
Stubborn set any other marking s: $T_s = T$.

Exemple Stubborn Set





Ex:

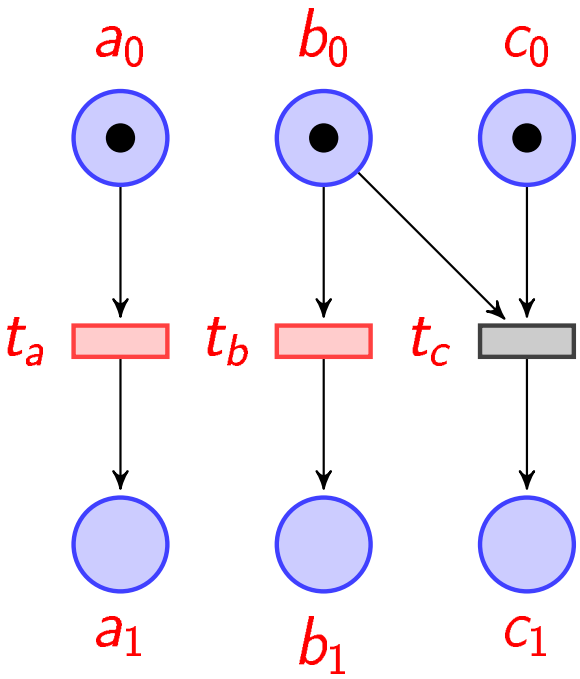
Stubborn set for marking 000: $T_{000} = \{t_a, t_b\}$.

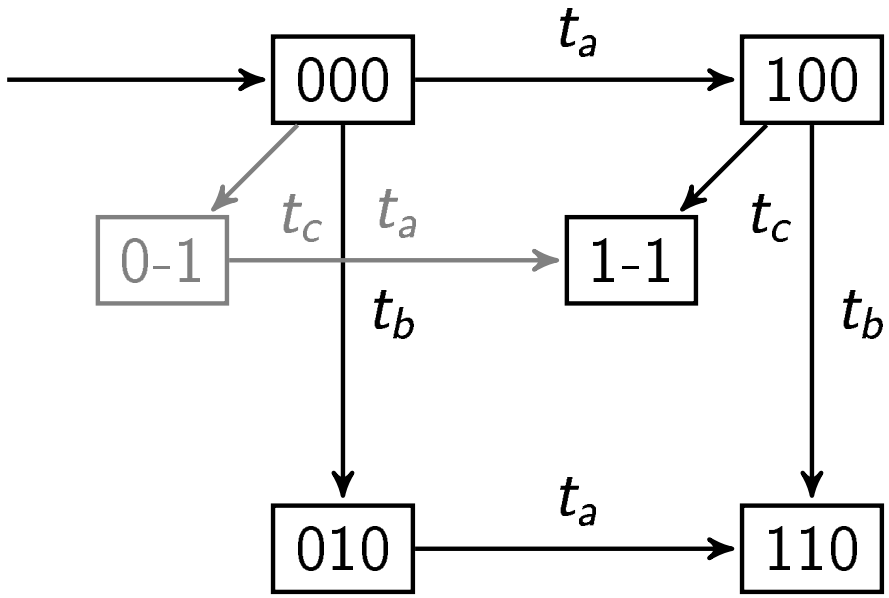
Stubborn set any other marking s: $T_s = T$.

Was it OK to set $T_{000} = \{t_a\}$?

$T_{000} = \{t_b\}$? $T_{000} = \{t_b, t_c\}$?

Exemple Stubborn Set





Ex:

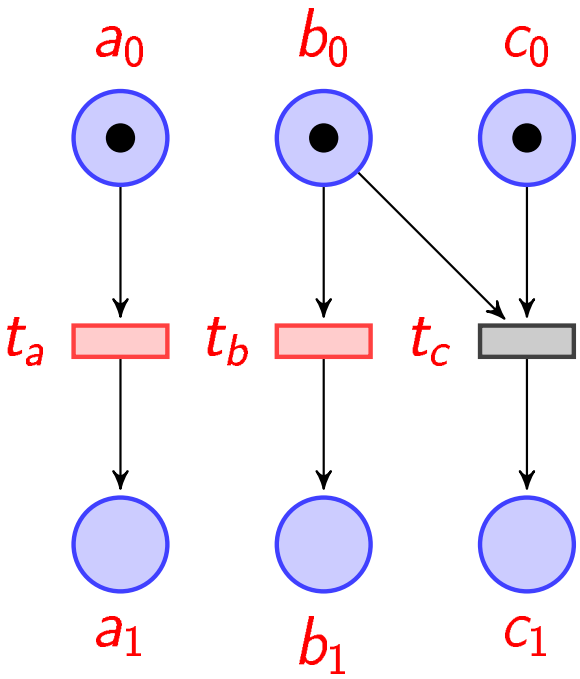
Stubborn set for marking 000: $T_{000} = \{t_a, t_b\}$.

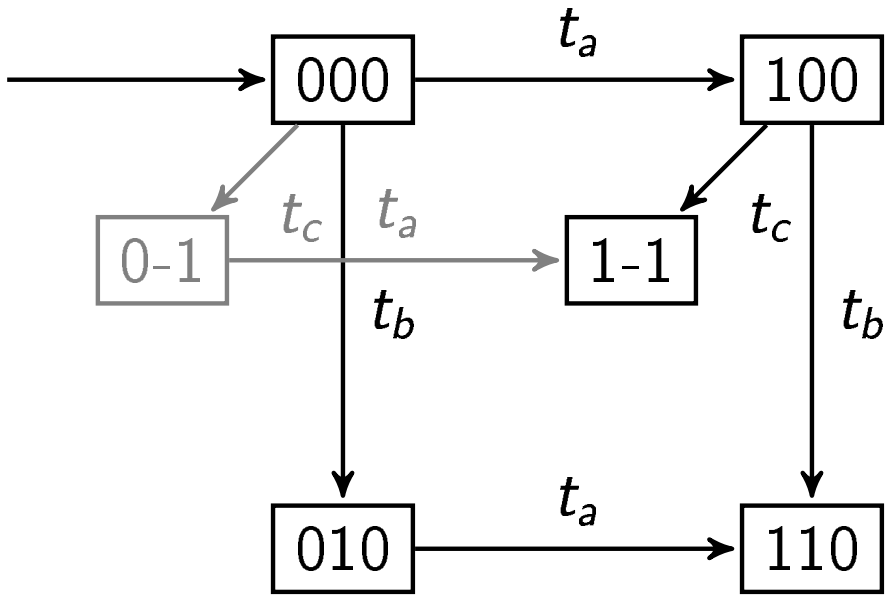
Stubborn set any other marking s: $T_s = T$.

Was it OK to set $T_{000} = \{t_a\}$? Yes.

$T_{000} = \{t_b\}$? $T_{000} = \{t_b, t_c\}$?

Exemple Stubborn Set





Ex:

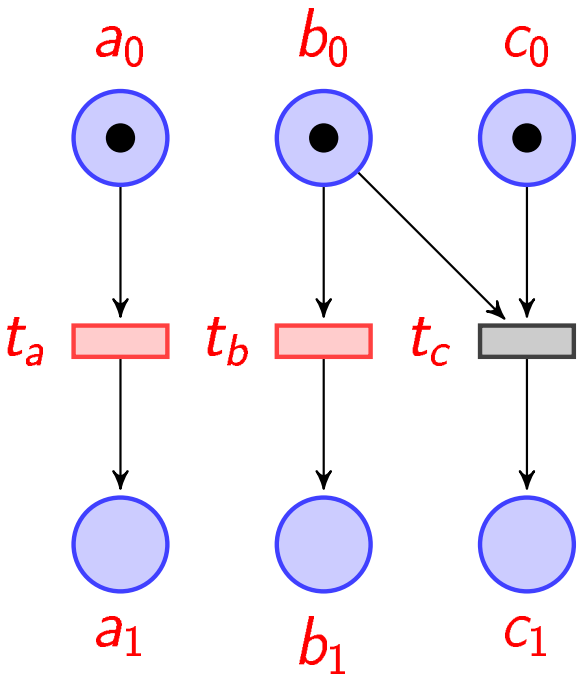
Stubborn set for marking 000: $T_{000} = \{t_a, t_b\}$.

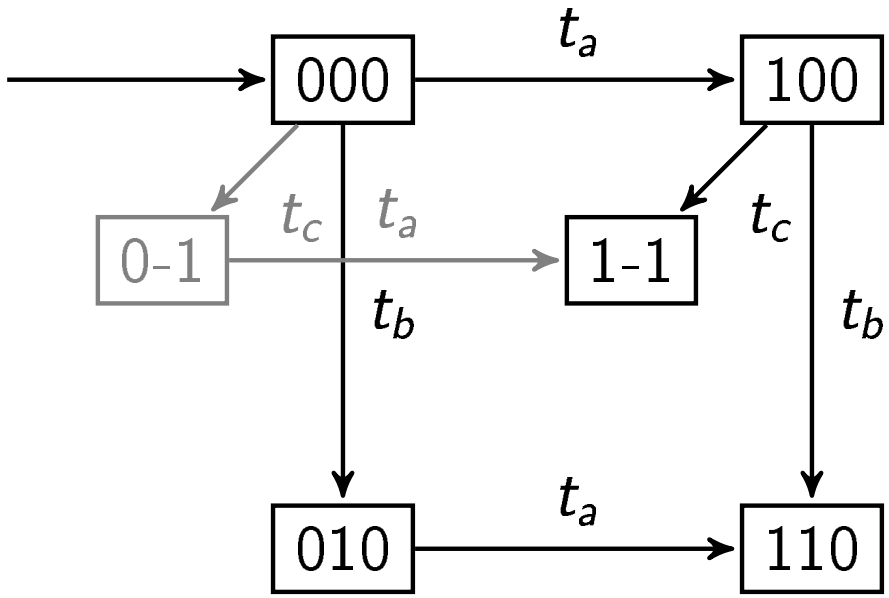
Stubborn set any other marking s: $T_s = T$.

Was it OK to set $T_{000} = \{t_a\}$? Yes.

$T_{000} = \{t_b\}$? No. $T_{000} = \{t_b, t_c\}$?

Exemple Stubborn Set





Ex:

Stubborn set for marking 000: $T_{000} = \{t_a, t_b\}$.

Stubborn set any other marking s: $T_s = T$.

Was it OK to set $T_{000} = \{t_a\}$? Yes.

$T_{000} = \{t_b\}$? No. $T_{000} = \{t_b, t_c\}$? Yes.

Computing Stubborn Sets

An abbreviation: $\Delta(t,s) = W(t,s) - W(s,t)$.

One way to compute a stubborn set for a non-dead marking M:

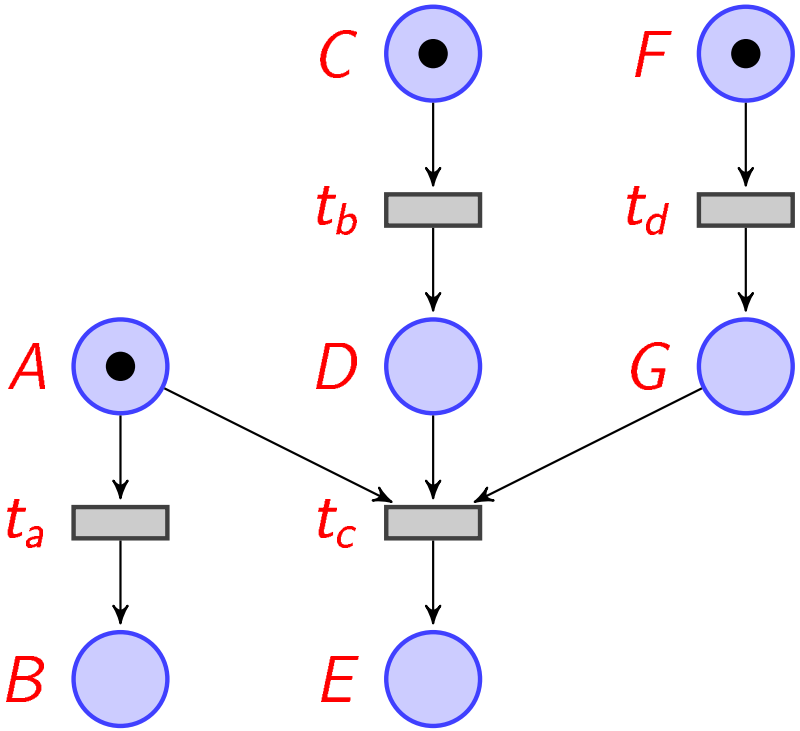
1. Pick a transition t enabled in M (i.e. $M \xrightarrow{t}$) and set $T_M = \{t\}$.
2. For any transition t in (and later added to) T_M:

   If $M \xrightarrow{t}$ Add to T_M any transition that can disable t.

   If $\neg M \xrightarrow{t}$ Pick a place $s \in {\bullet t}$ so that $M(s) < \Delta(t,s)$.
   Add to T_M all transitions r so that $\Delta(r,s) > 0$.

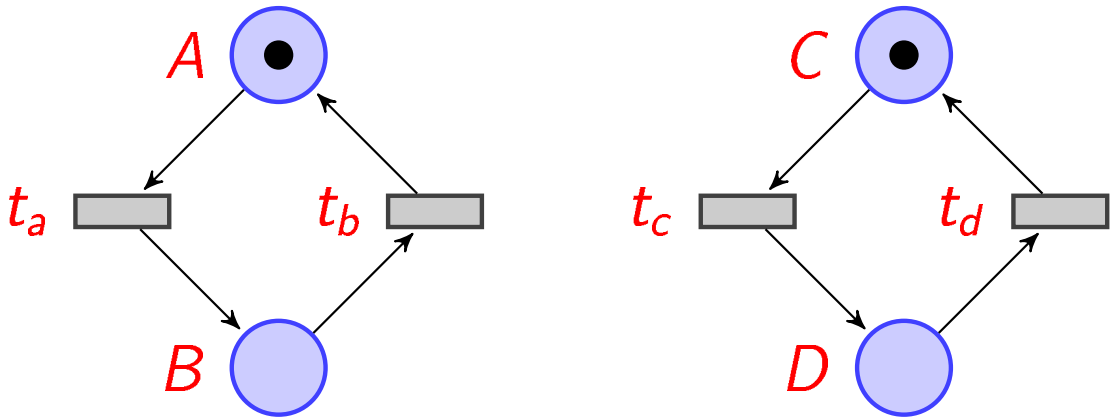
The set of all enabled transitions is always a valid stubborn set, so we should never try to build a larger set.

Example



Ignoring problem

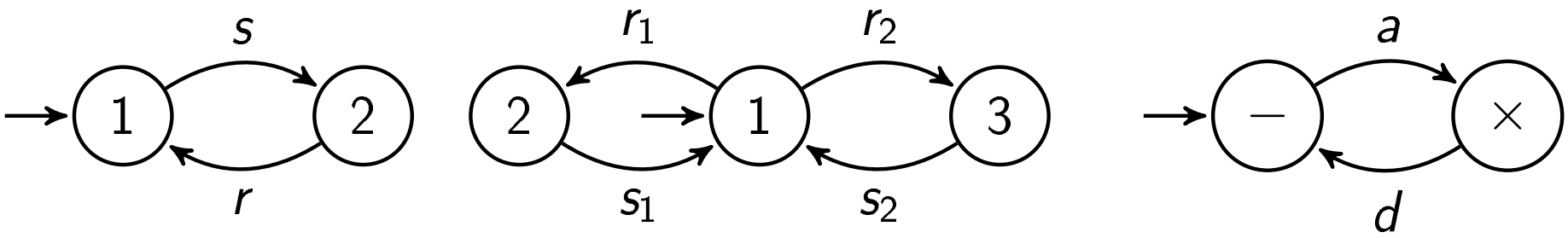
Some words about about verifying infinite behaviors.



$T_{AC} = \{t_a\}$ and $T_B = \{t_b\}$ implies that $\{t_c, t_d\}$ are never fired.

We also need to deal with properties.

Client/Server using Synchronized Automata

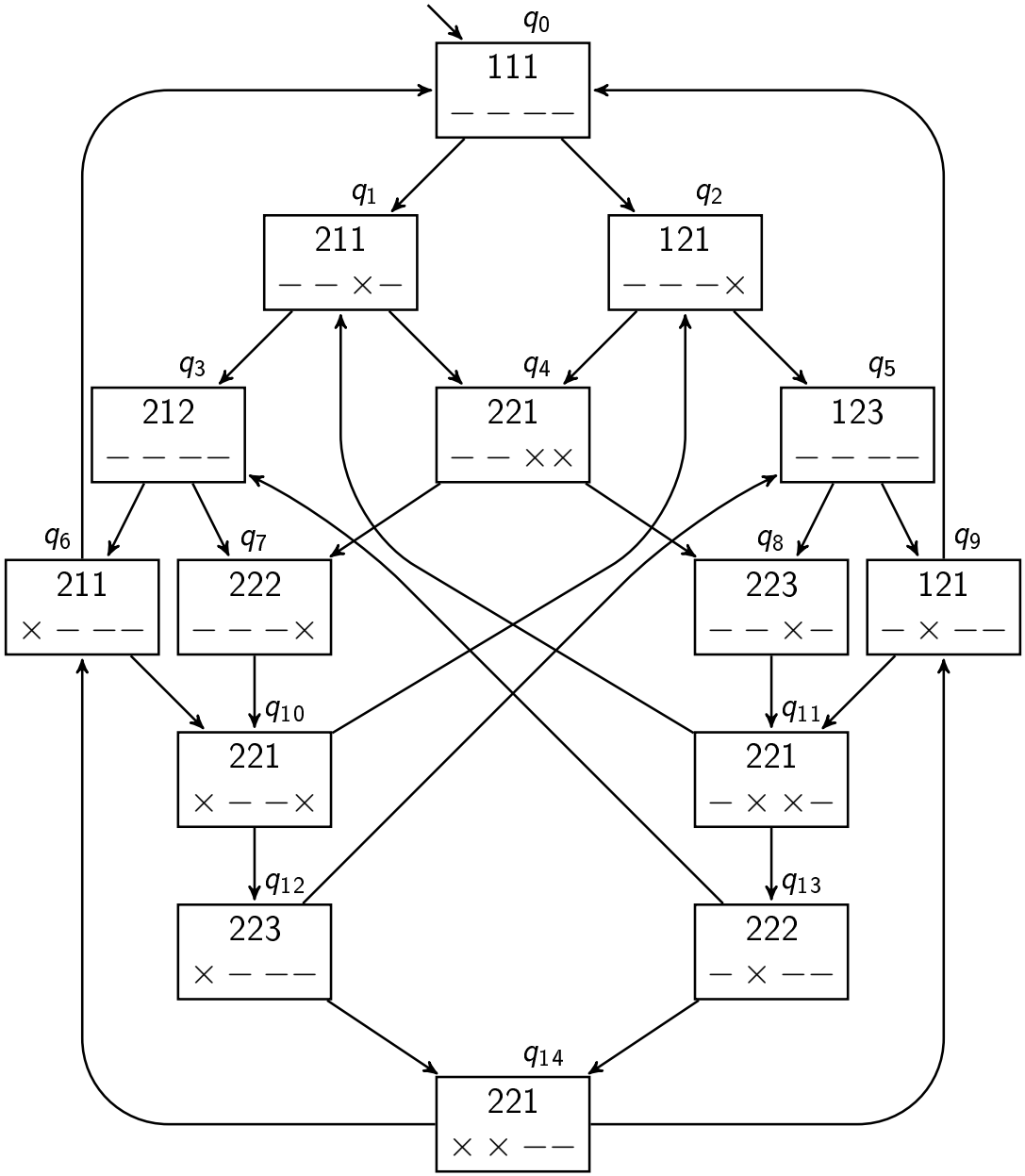


Client C Server S Canal B

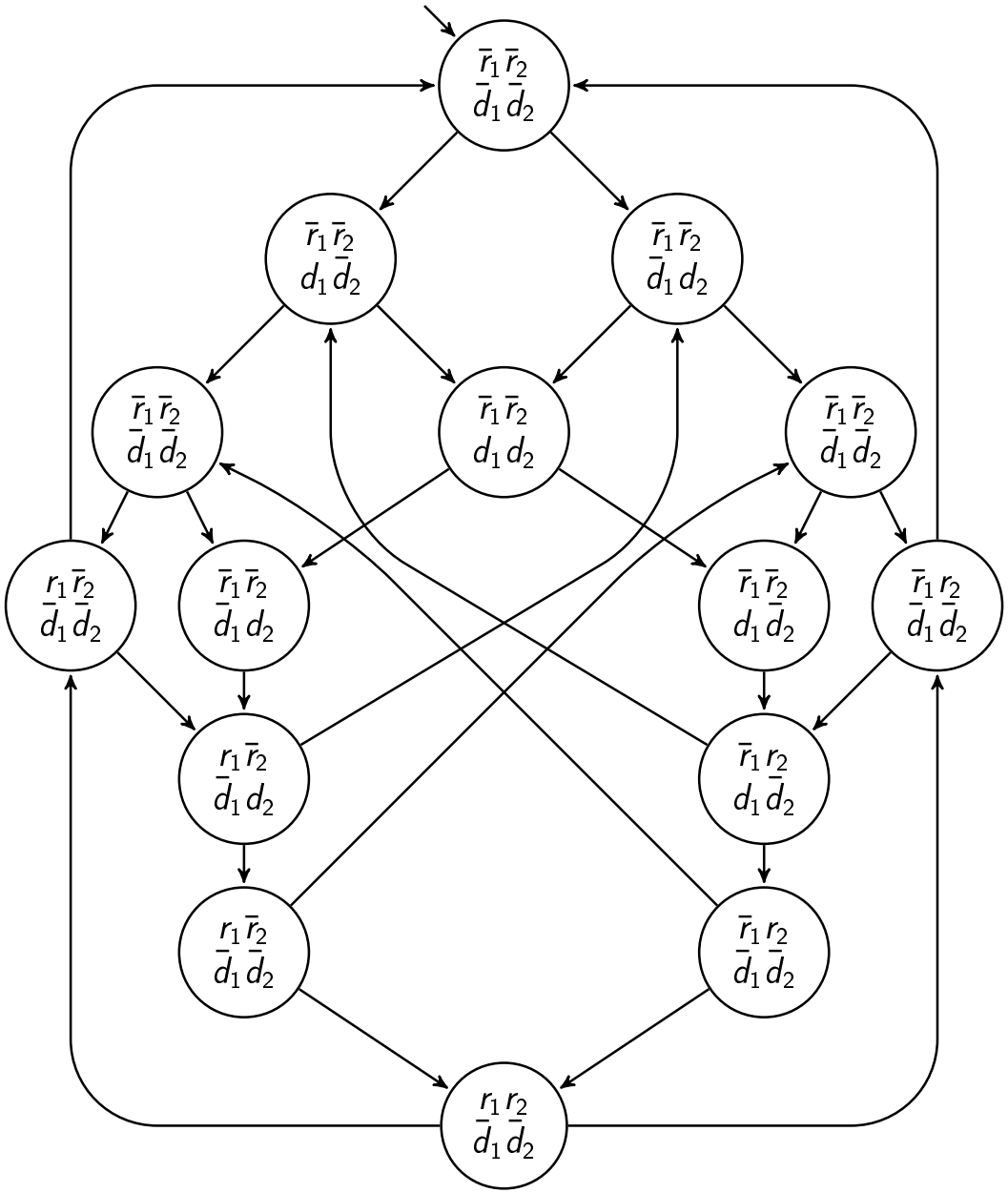
Synchronization rule for system $\langle C, C, S, B, B, B, B \rangle$:

$$
\begin{array}{ll}
(1) & \langle\ s\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ ,\ a\ ,\ .\ \rangle \\
(2) & \langle\ .\ ,\ s\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ ,\ a\ \rangle \\
(3) & \langle\ r\ ,\ .\ ,\ .\ ,\ .\ ,\ d\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ \rangle \\
(4) & \langle\ .\ ,\ r\ ,\ .\ ,\ .\ ,\ .\ ,\ d\ ,\ .\ ,\ .\ ,\ .\ \rangle \\
(5) & \langle\ .\ ,\ .\ ,\ .\ ,\ r_1\ ,\ .\ ,\ .\ ,\ .\ ,\ d\ ,\ .\ \rangle \\
(6) & \langle\ .\ ,\ .\ ,\ .\ ,\ s_1\ ,\ a\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ \rangle \\
(7) & \langle\ .\ ,\ .\ ,\ .\ ,\ r_2\ ,\ .\ ,\ .\ ,\ .\ ,\ .\ ,\ d\ \rangle \\
(8) & \langle\ .\ ,\ .\ ,\ .\ ,\ s_2\ ,\ .\ ,\ a\ ,\ .\ ,\ .\ ,\ .\ \rangle
\end{array}
$$

State Space


q_0
111
– – ––
q_1
211
– – ×–
q_2
121
– – –×
q_3
212
– – ––
q_4
221
– – ××
q_5
123
– – ––
q_6
211
× – ––
q_7
222
– – –×
q_8
223
– – ×–
q_9
121
– × ––
q_{10}
221
× – –×
q_{11}
221
– × ×–
q_{12}
223
× – ––
q_{13}
222
– × ––
q_{14}
221
× × ––

Kripke Structure to Simplify



Kripke Structure to Simplify