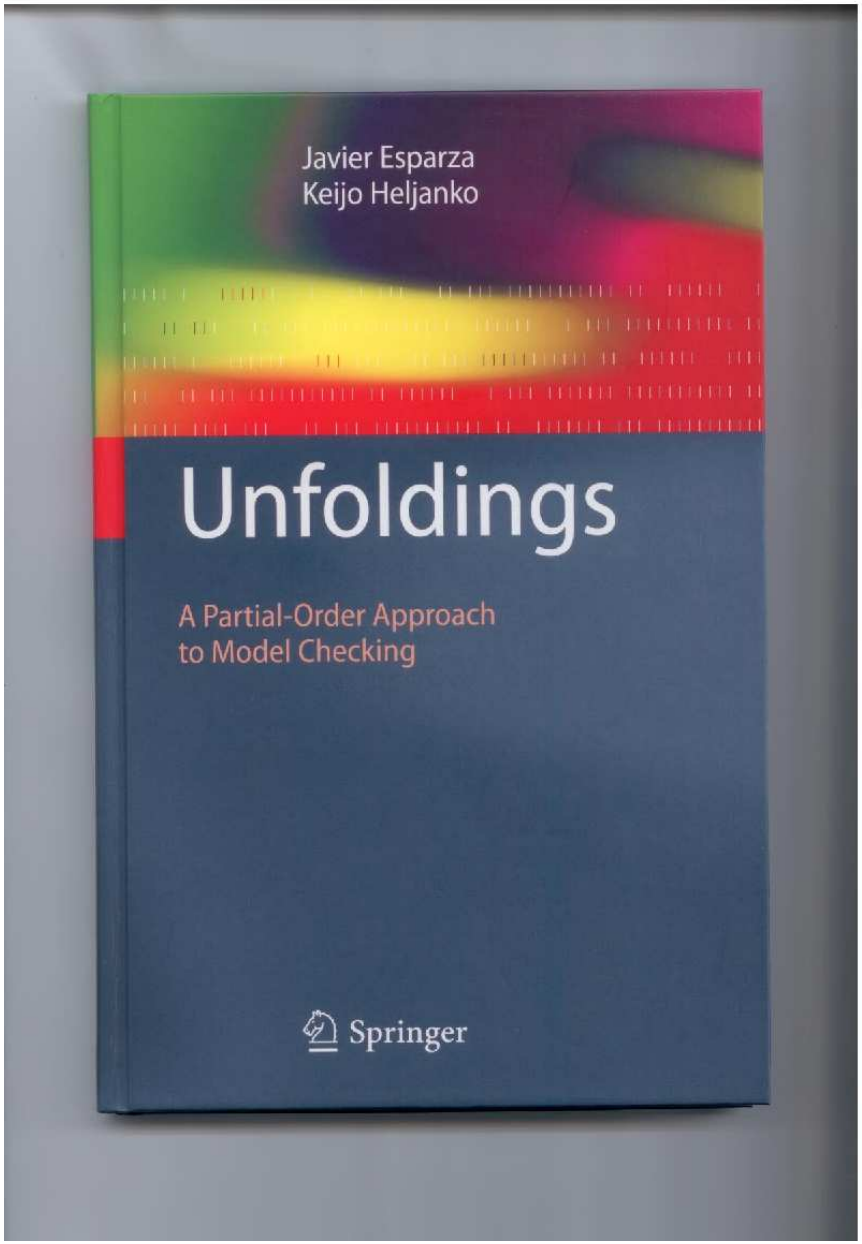
A False History of True Concurrency — from Petri to tools

Javier Esparza

Technische Universität München

2008


Javier Esparza
Keijo Heljanko
Unfoldings
A Partial-Order Approach
to Model Checking
Springer

1962

Abstract Models of Computation in the early 60s

Lambda calculus (Church 35)

Turing machines (Turing 36)

Finite automata (Kleene 56, Moore 56, Mealy 56, Scott and Rabin 59)

Pushdown automata (Oettinger 61, Chomsky 62)

Semantics: executions

Configurations: set of informations completely describing the state of the machine at a certain moment in time

Transitions: moves between configurations

Lambda calculus	$(\lambda x.xx)(\lambda y.y)$	$\longrightarrow$	$(\lambda y.y)(\lambda z.z)$
Turing machine	$0010q_1011$	$\longrightarrow$	$001q_201011$
Finite automaton	q_1	$\xrightarrow{a}$	q_2
Pushdown automaton	$(q_1, XYYZ)$	$\xrightarrow{a}$	$(q_2, XYXYYZ)$

Executions: alternating sequences of configurations and transitions

Physics and Computation

Abstract machines are implemented as physical machines

Physics and Computation

Abstract machines	are implemented as can simulate	physical machines

SIMULA project (Nygaard and Dahl), started in 1962

Physics and Computation

Abstract machines	are implemented as can simulate	physical machines

SIMULA project (Nygaard and Dahl), started in 1962

A plane (physical machine)

Physics and Computation

Abstract machines	are implemented as can simulate	physical machines

SIMULA project (Nygaard and Dahl), started in 1962

A plane (physical machine)

can be simulated by a flight simulator (abstract machine)

Physics and Computation

Abstract machines	are implemented as can simulate	physical machines

SIMULA project (Nygaard and Dahl), started in 1962

A plane (physical machine)

can be simulated by a flight simulator (abstract machine)

which can be implemented in a video console (physical machine)

Physics and Computation

Abstract machines	are implemented as can simulate	physical machines

SIMULA project (Nygaard and Dahl), started in 1962

A plane (physical machine)

can be simulated by a flight simulator (abstract machine)

which can be implemented in a video console (physical machine)

which can be simulated by a hardware simulator (abstract machine)

Physics and Computation

Abstract machines	are implemented as can simulate	physical machines

SIMULA project (Nygaard and Dahl), started in 1962

A plane (physical machine)

can be simulated by a flight simulator (abstract machine)

which can be implemented in a video console (physical machine)

which can be simulated by a hardware simulator (abstract machine)

which is implemented in a PC (physical machine) ...

Petri's question

In 1962, C.A. Petri (1926–2010) points out a discrepancy between how Theoretical Physics and Theoretical Computer Science describe systems (at that time):

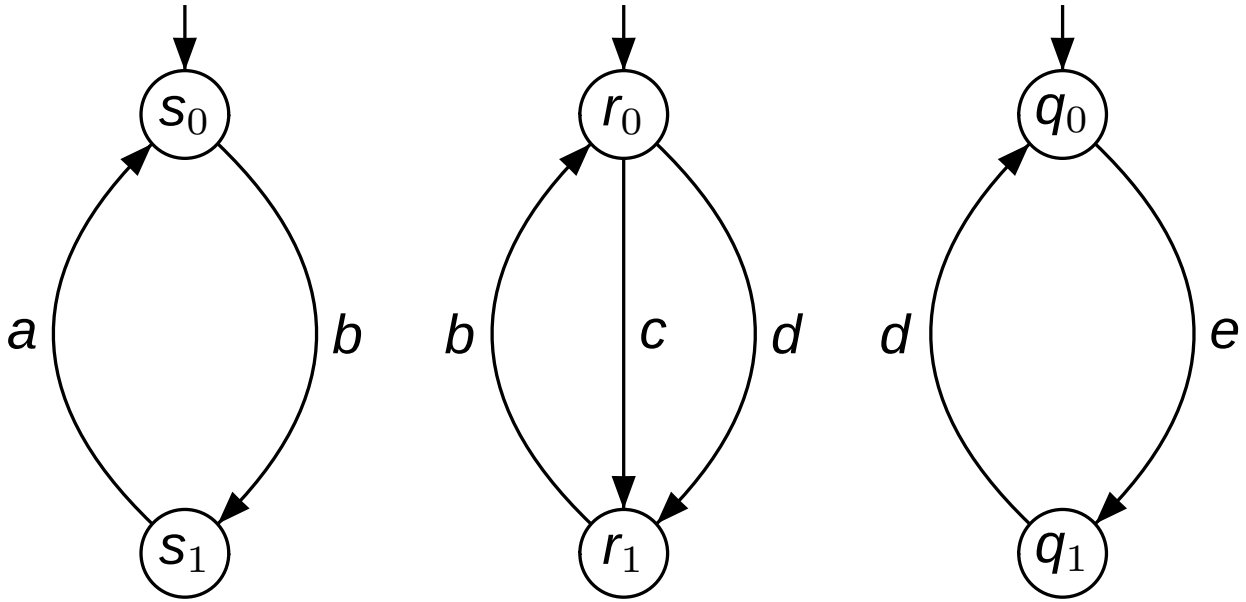
> Theoretical Physics describes systems as a collection of interacting particles (subsystems), without a notion of global clock or simultaneity
>
> Theoretical Computer Science describes systems as sequential virtual machines going through a temporally ordered sequence of global states

Petri's question:

> Which kind of abstract machine should be used to describe the physical implementation of a Turing machine?

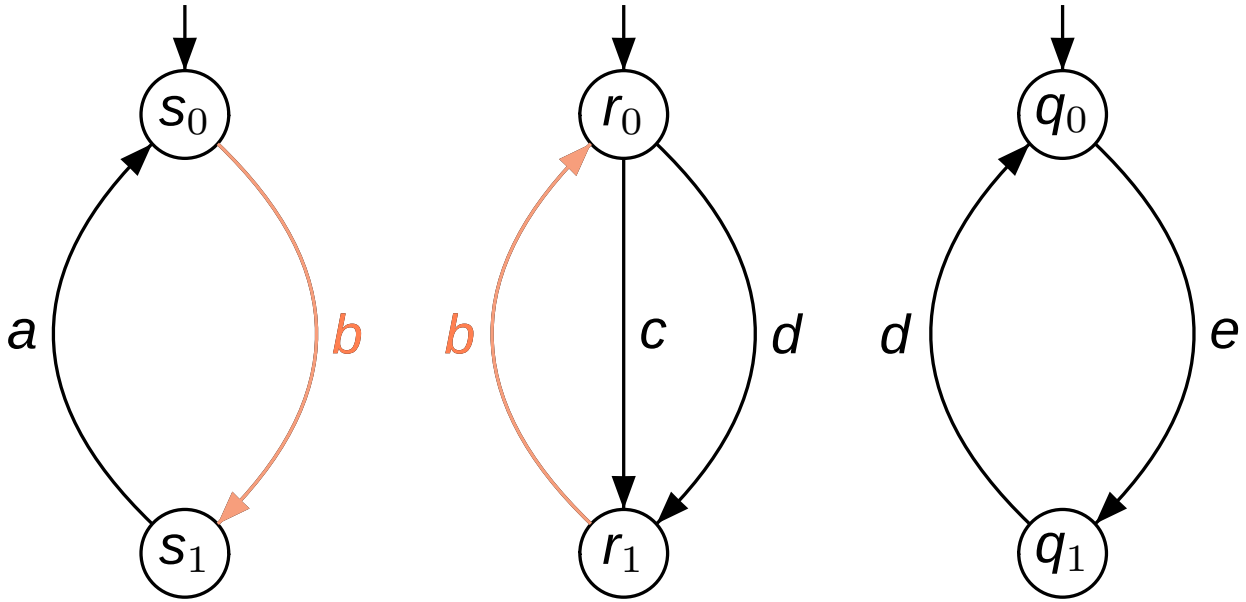
Petri Nets

A graphical representation of interacting finite automata:



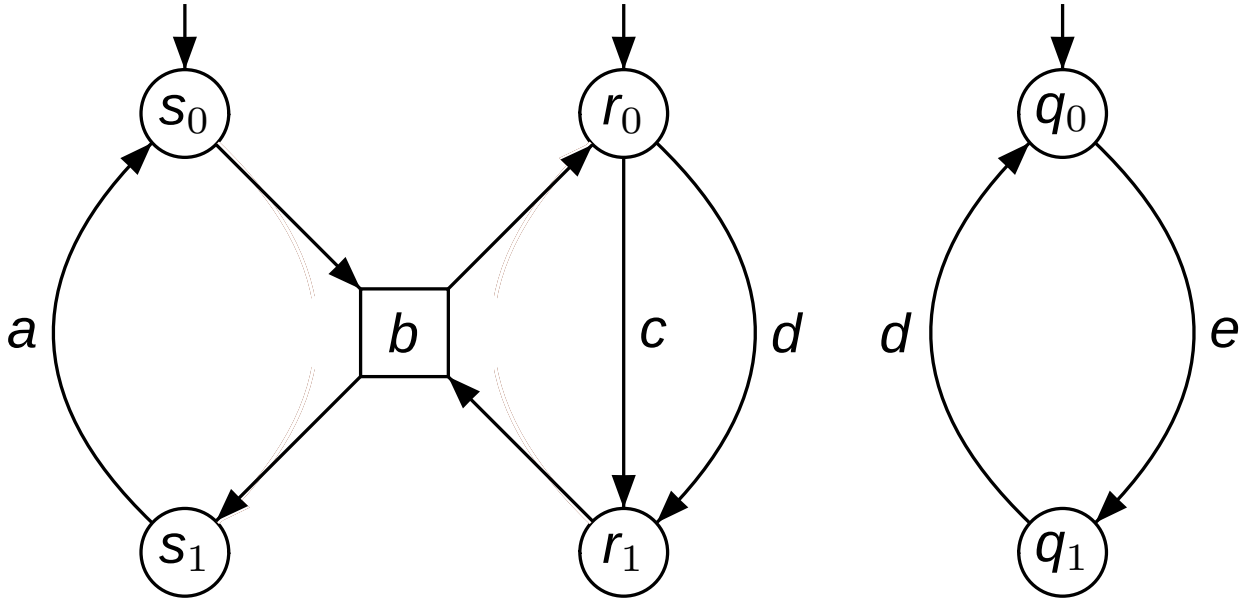
Petri Nets

A graphical representation of interacting finite automata:



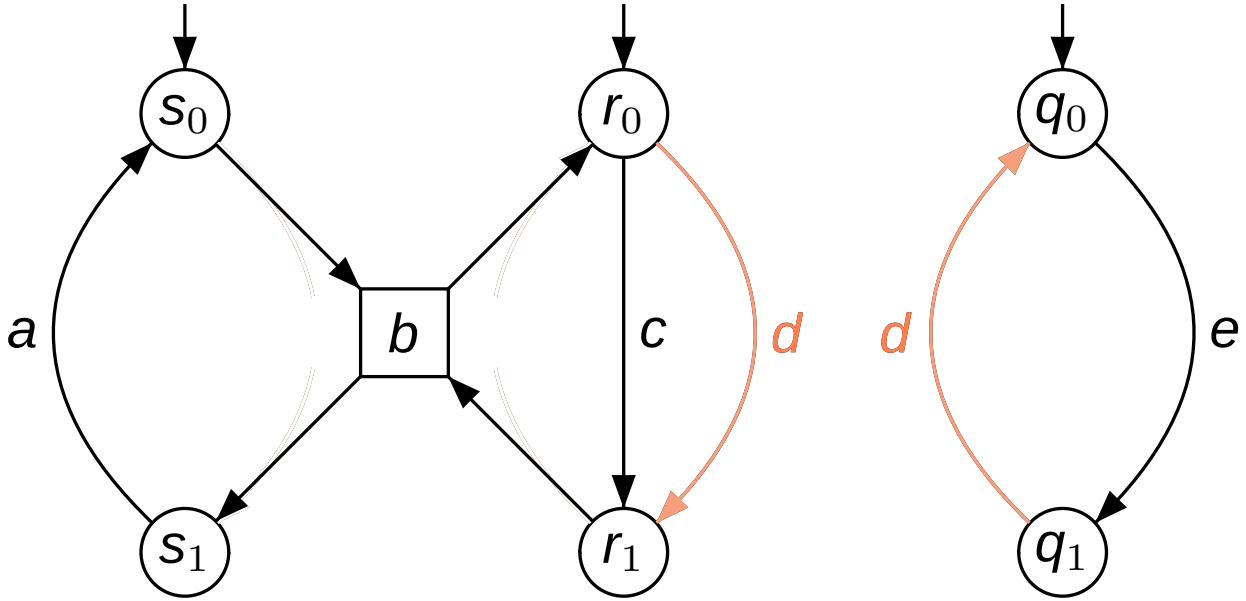
Petri Nets

A graphical representation of interacting finite automata:



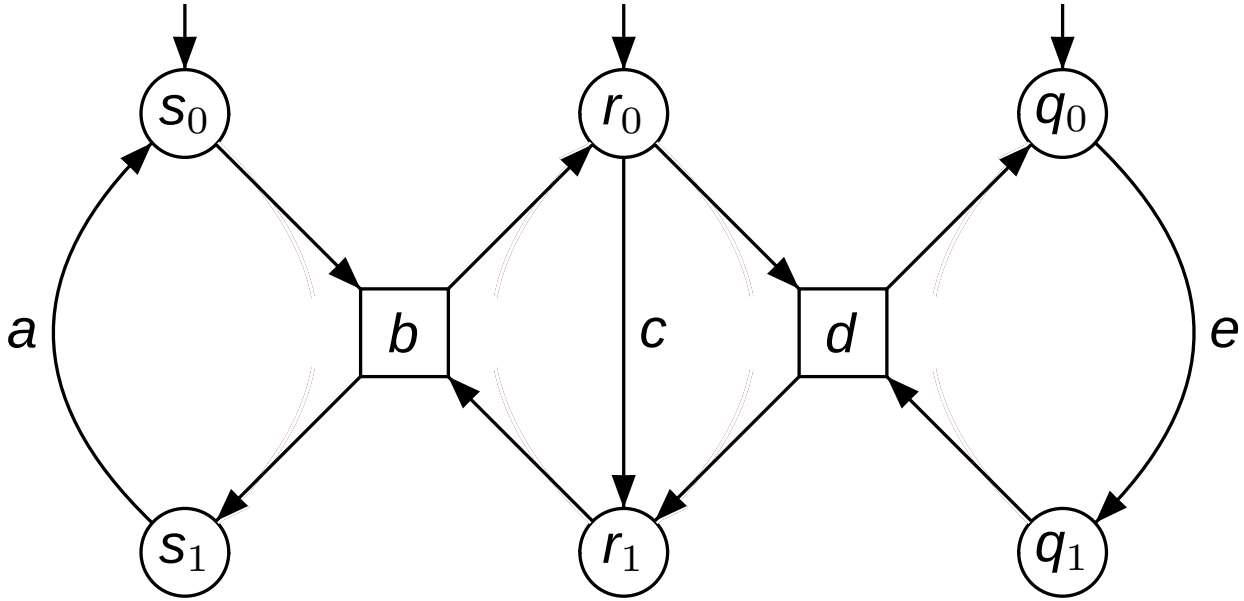
Petri Nets

A graphical representation of interacting finite automata:



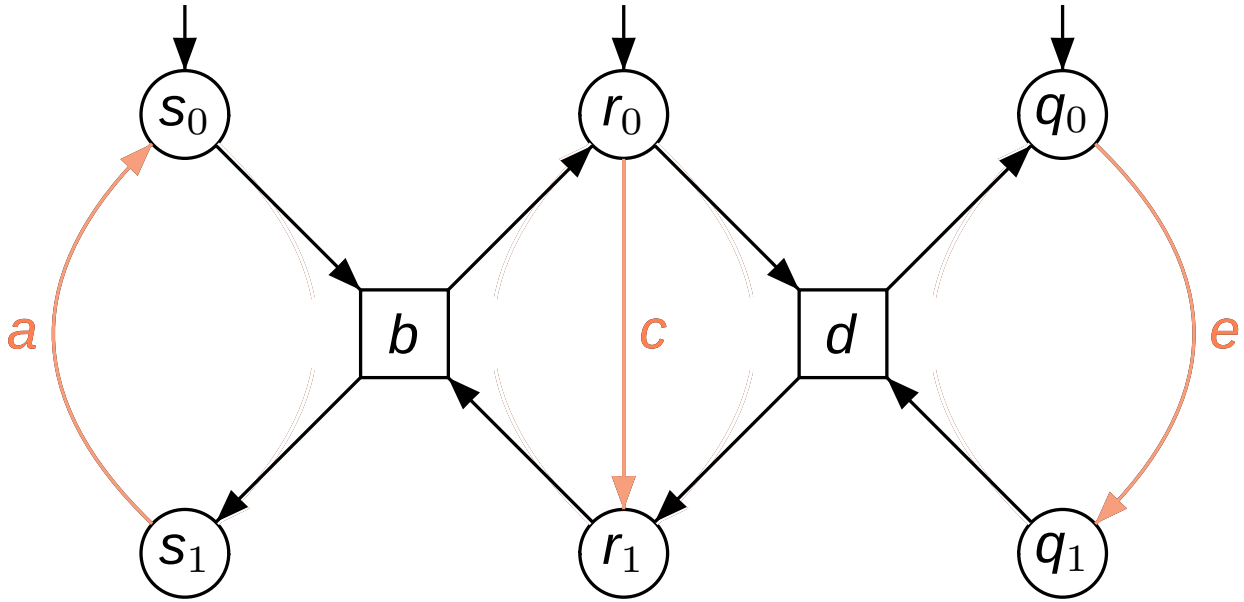
Petri Nets

A graphical representation of interacting finite automata:



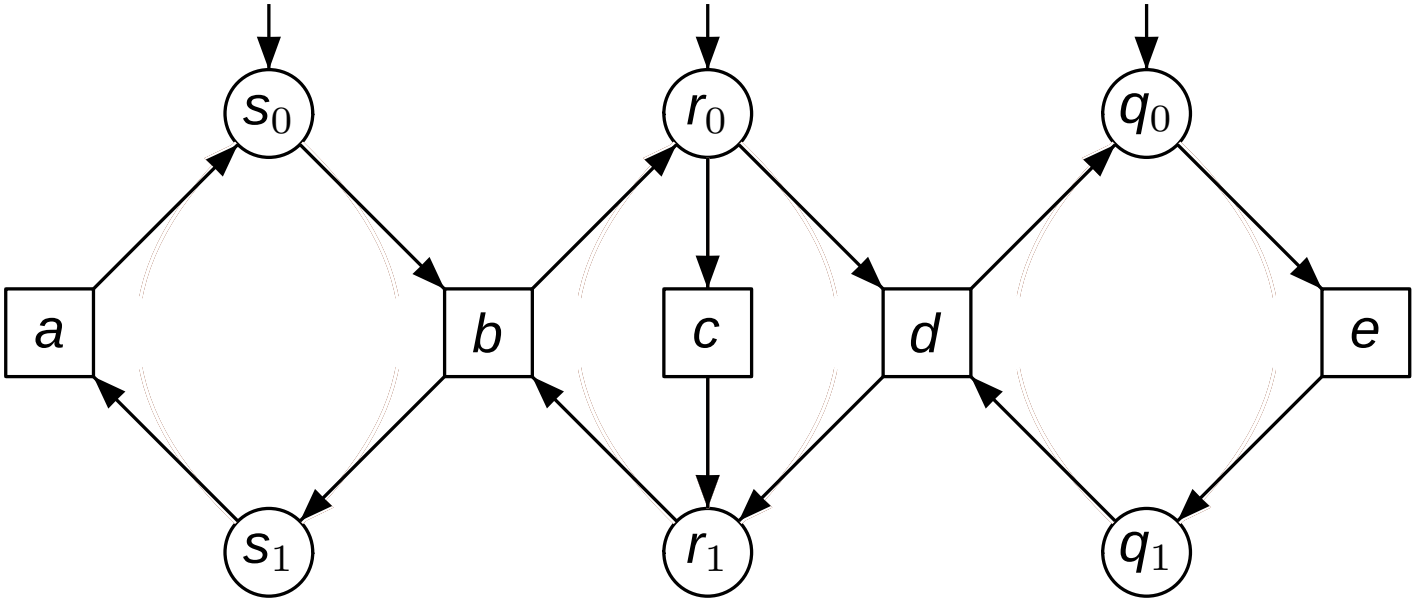
Petri Nets

A graphical representation of interacting finite automata:



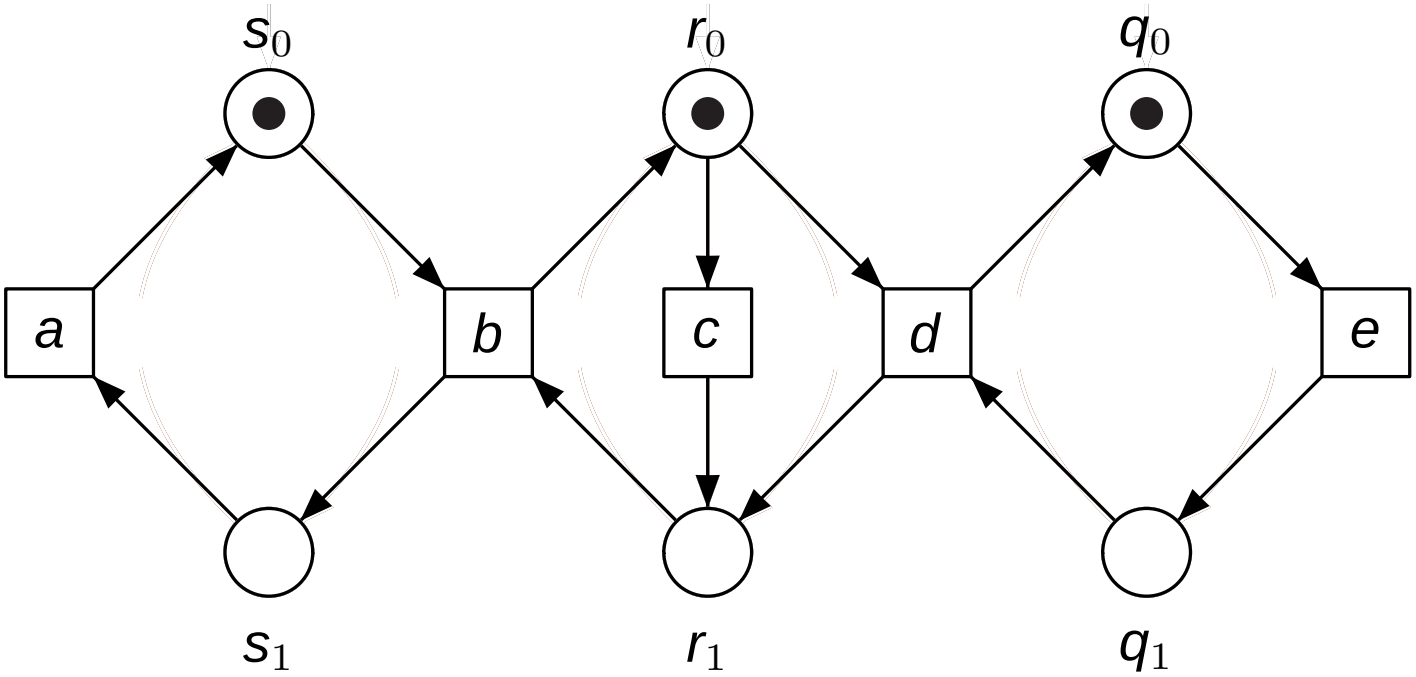
Petri Nets

A graphical representation of interacting finite automata:



Petri Nets

A graphical representation of interacting finite automata:



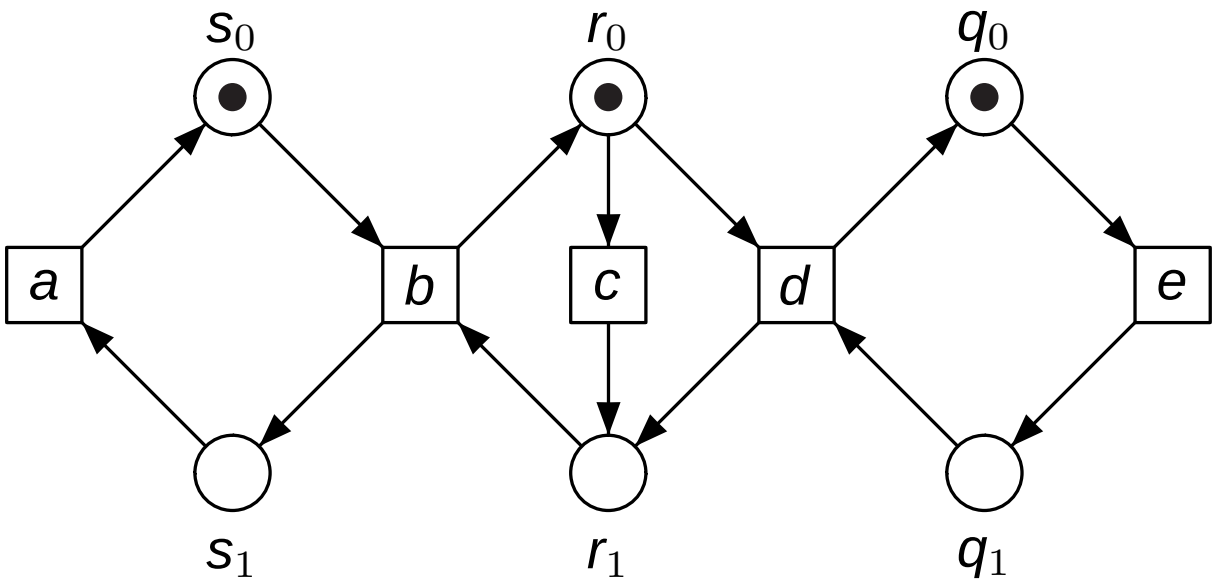
Execution semantics

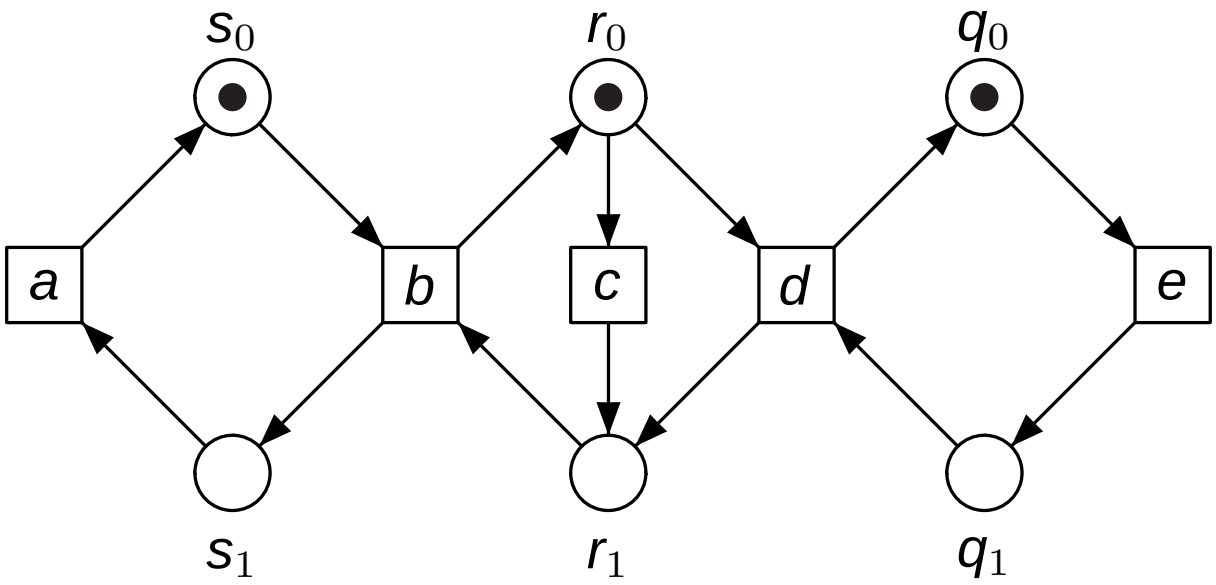
An execution semantics

State: marking (distribution of tokens)

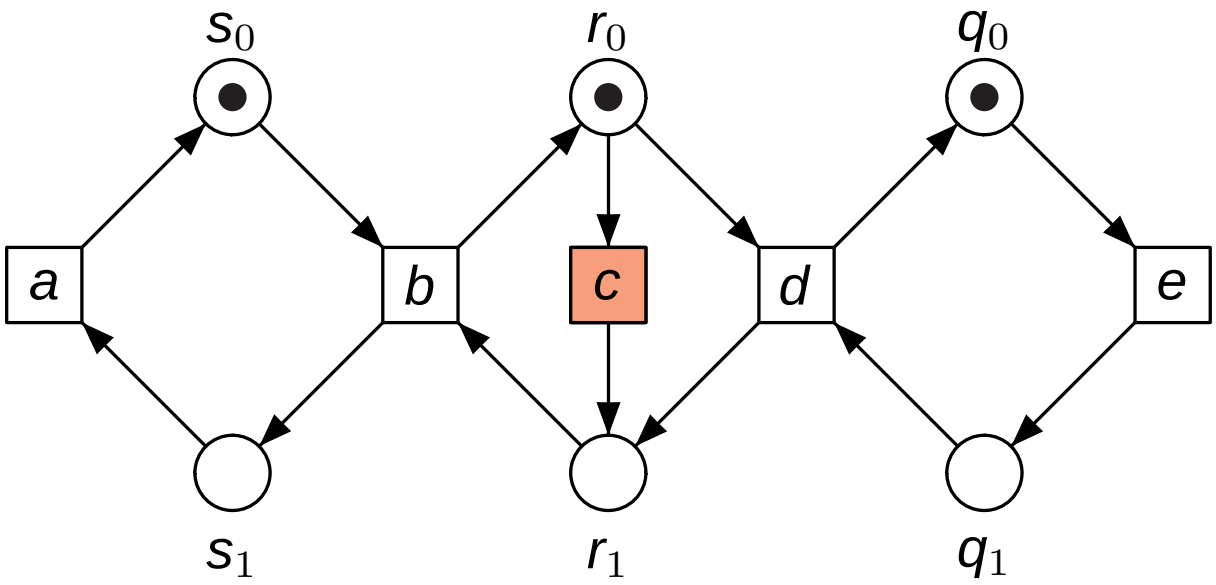
Transitions: $M \xrightarrow{a} M'$

Executions: $M_0 \xrightarrow{a_0} M_1 \xrightarrow{a_1} M_2 \ldots$

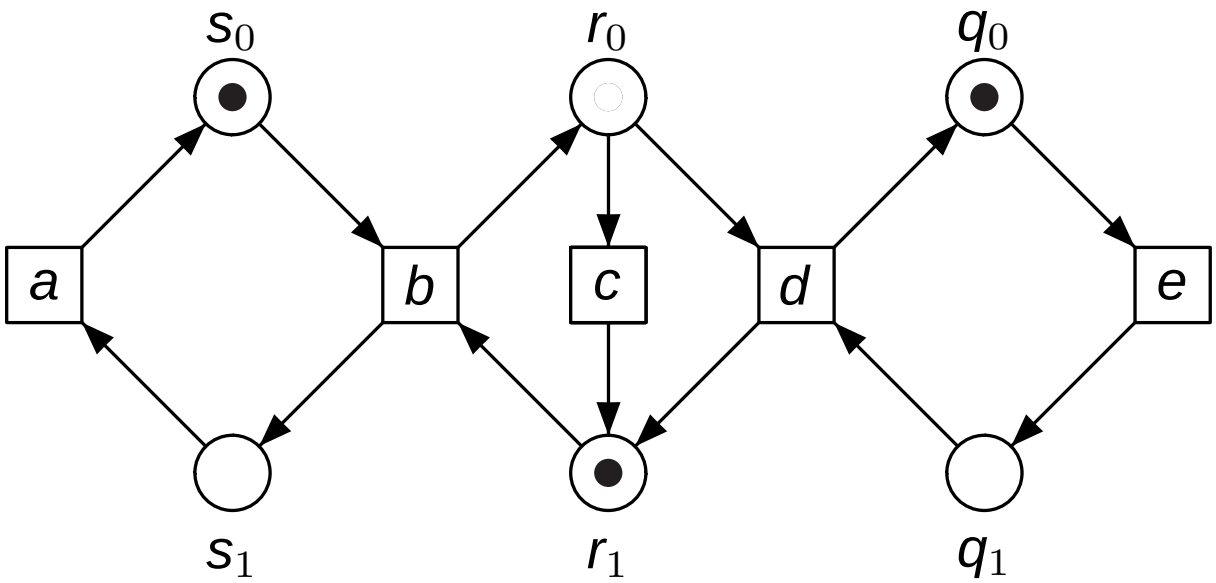

s_0
r_0
q_0
a
b
c
d
e
s_1
r_1
q_1



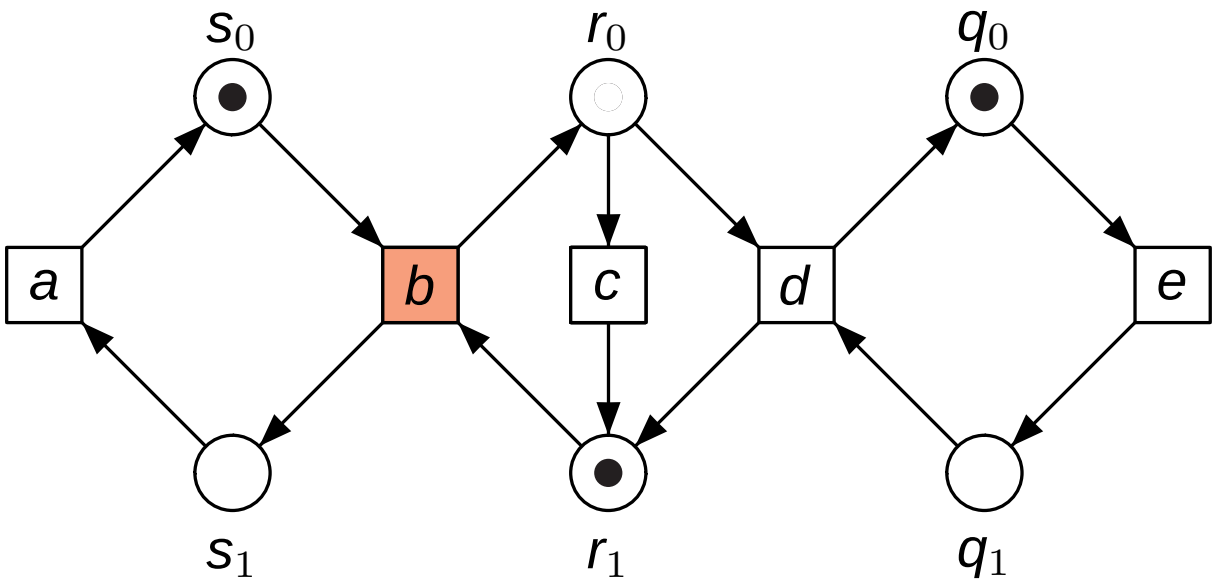
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$



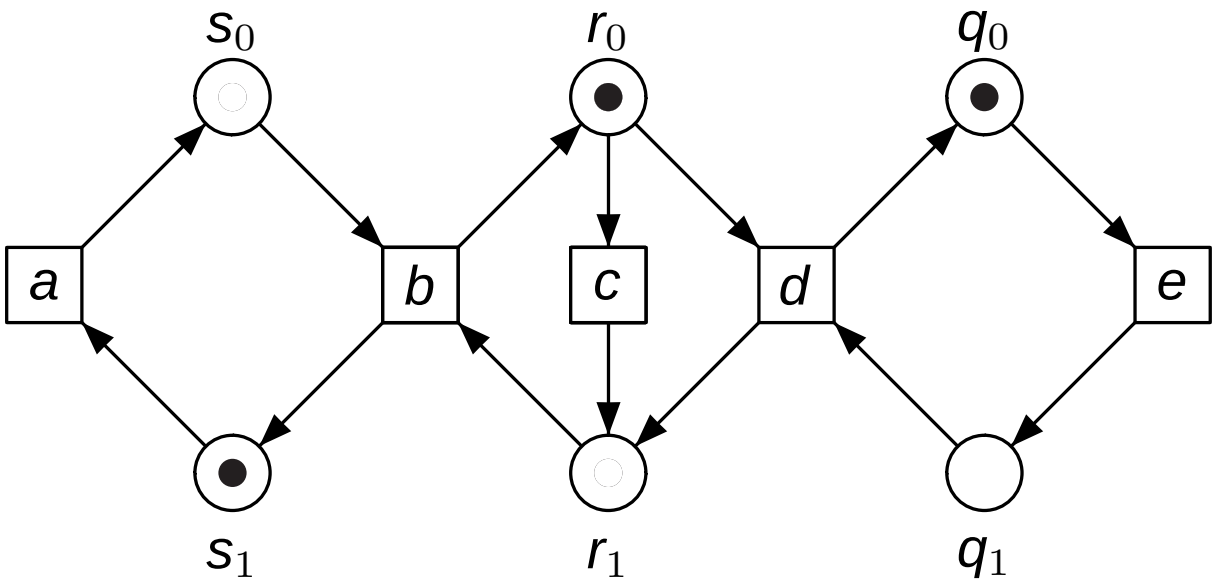
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$



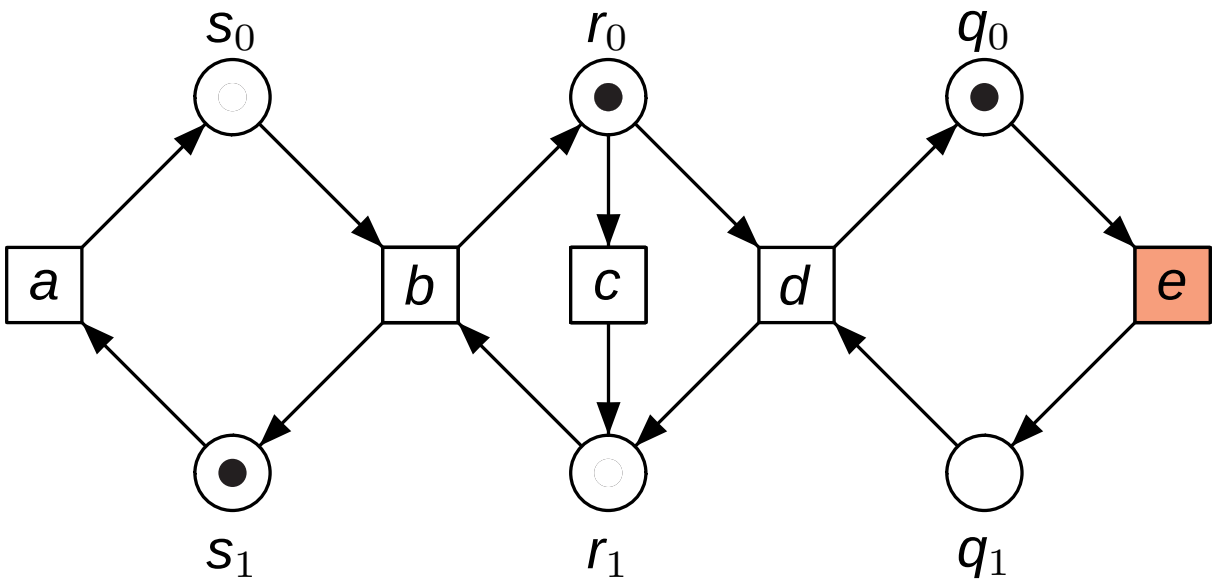
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



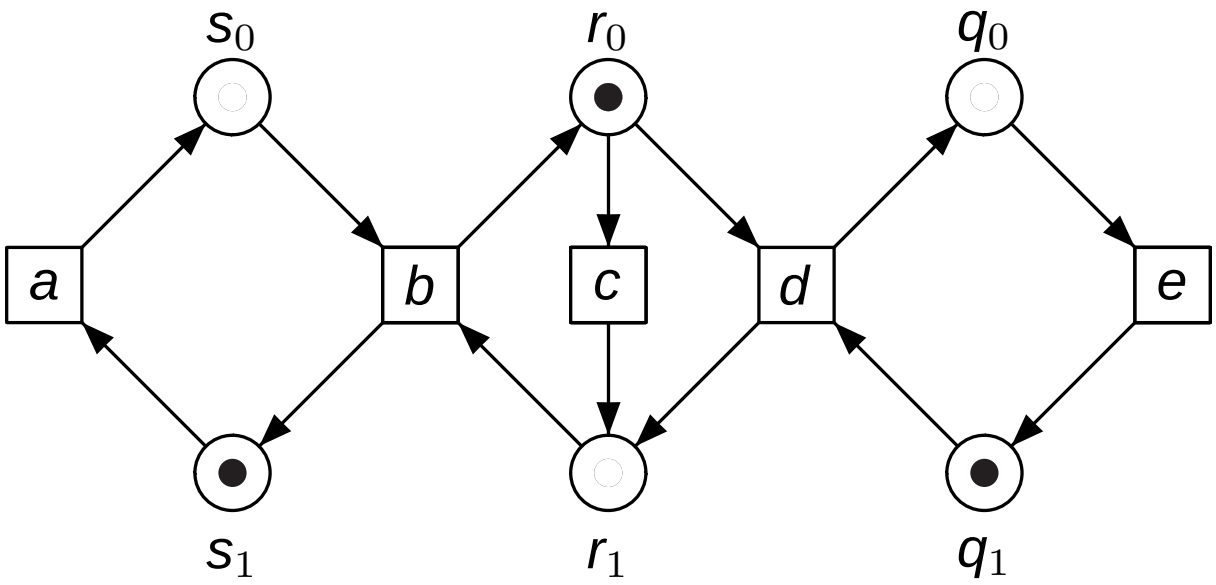
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$



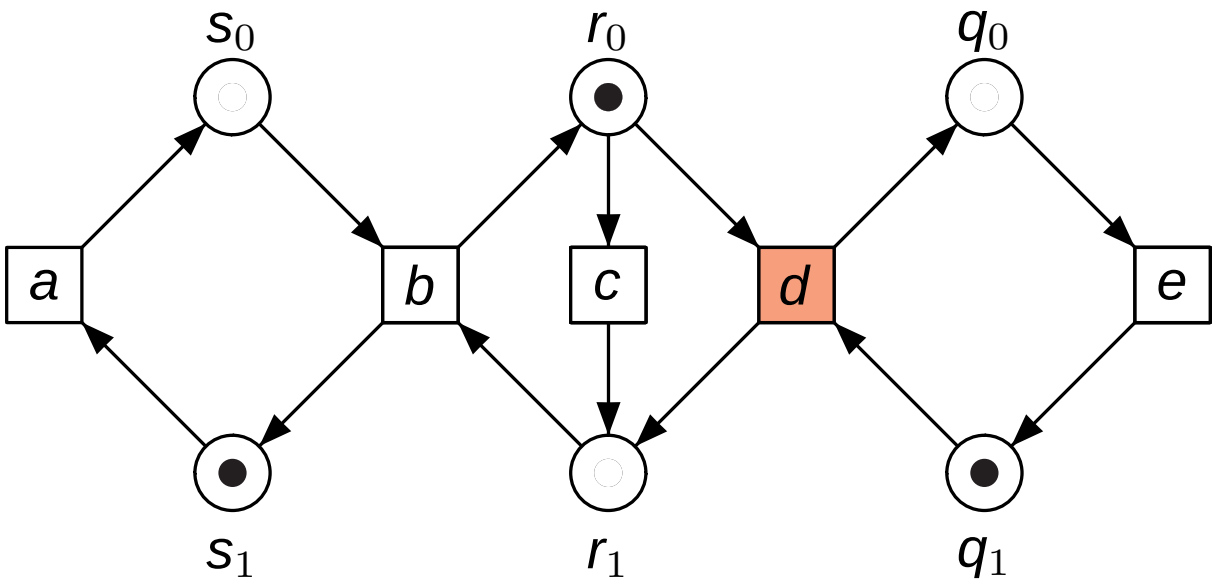
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{b} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



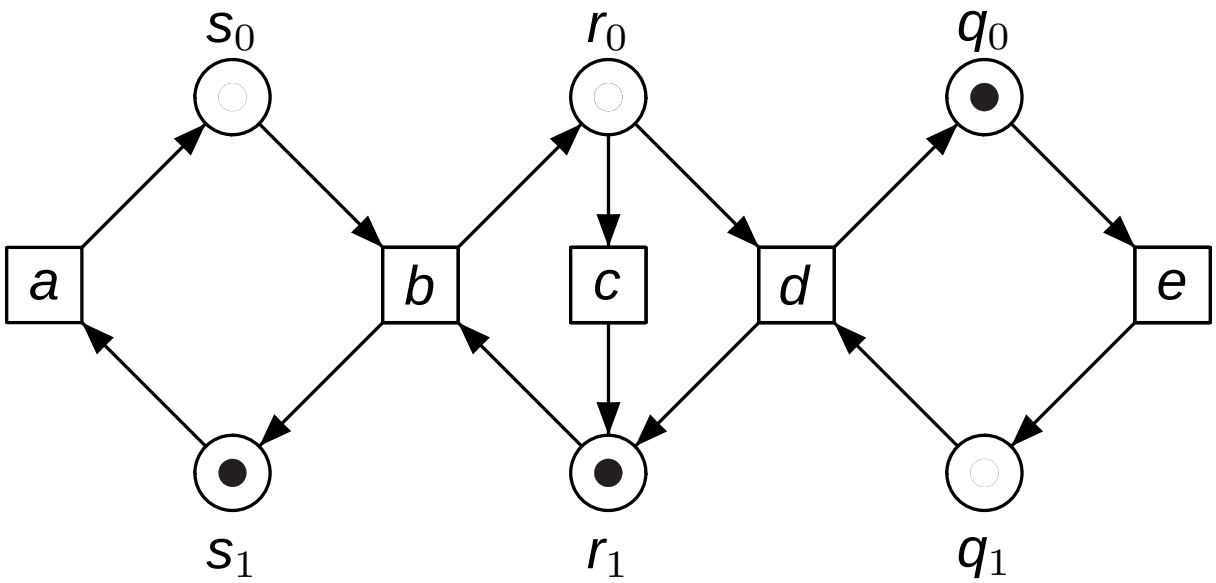
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{b} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$



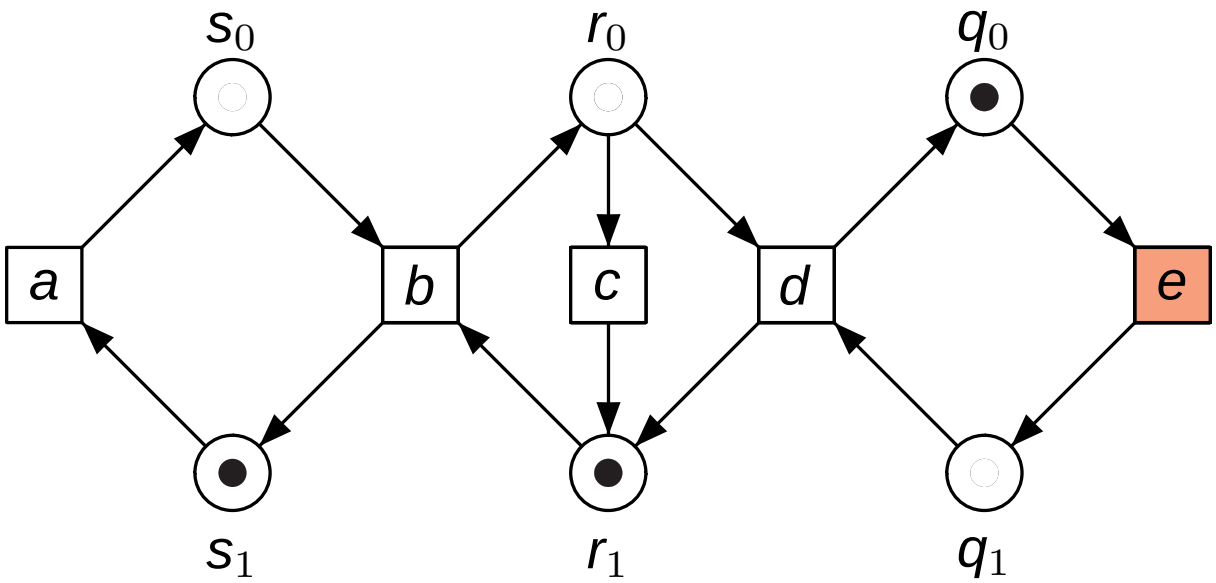
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{b} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{e} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$



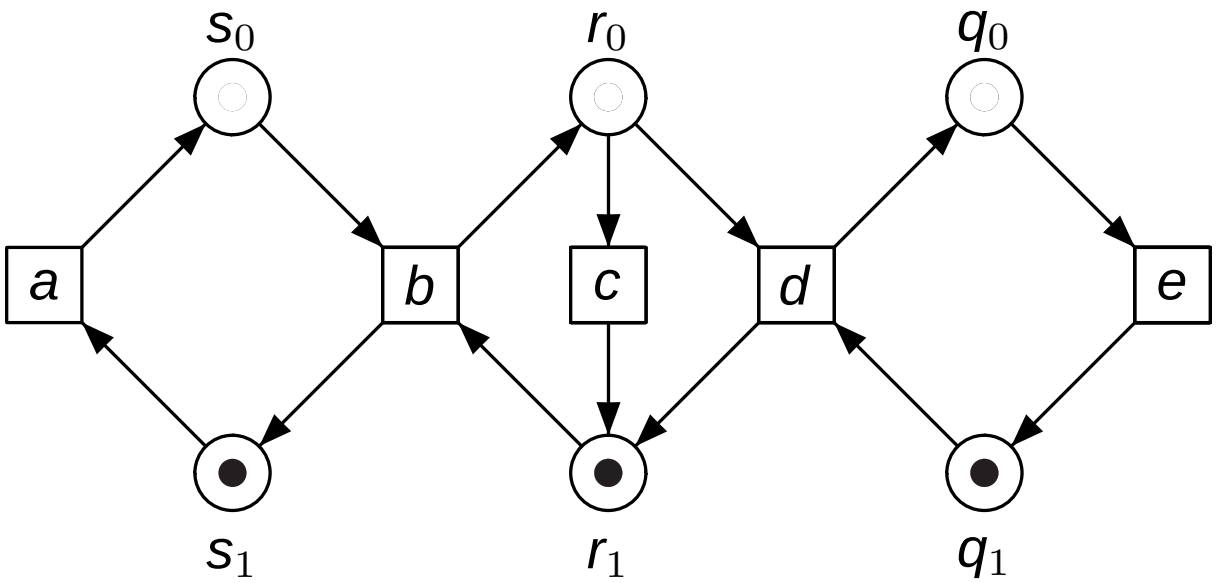
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{b} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{e} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$



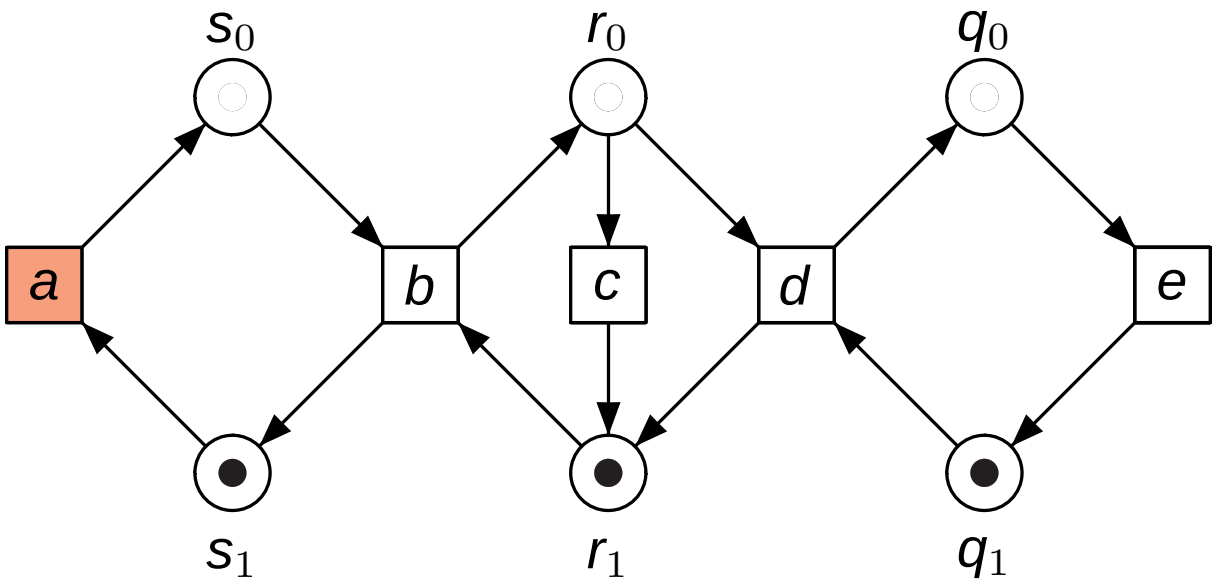
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{b} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{e} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \xrightarrow{d} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$



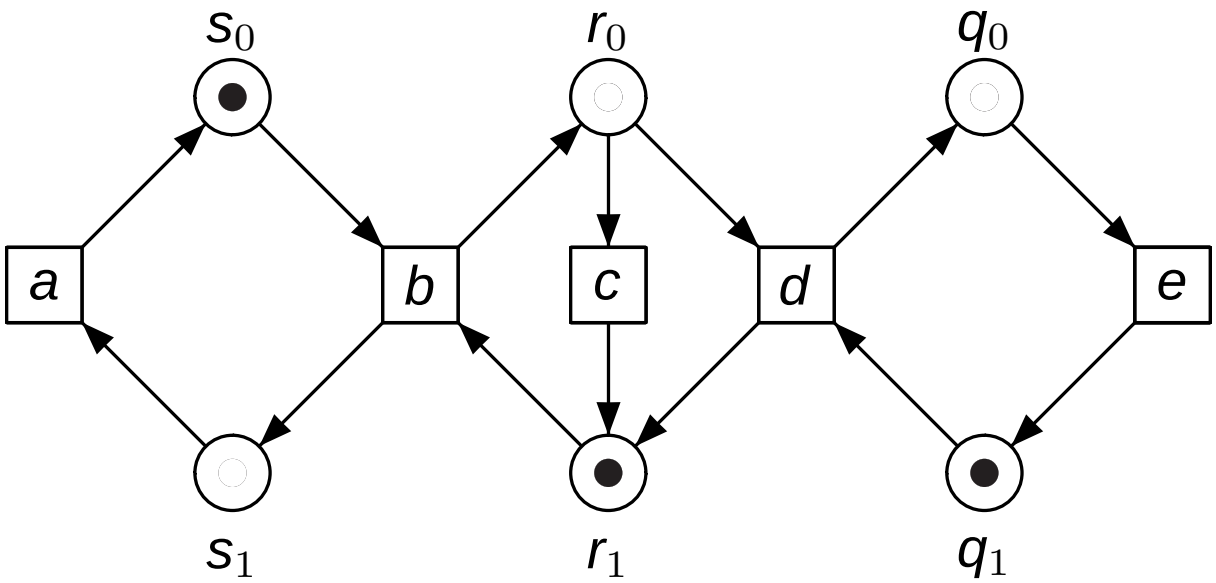
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{b} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{e} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \xrightarrow{d} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$



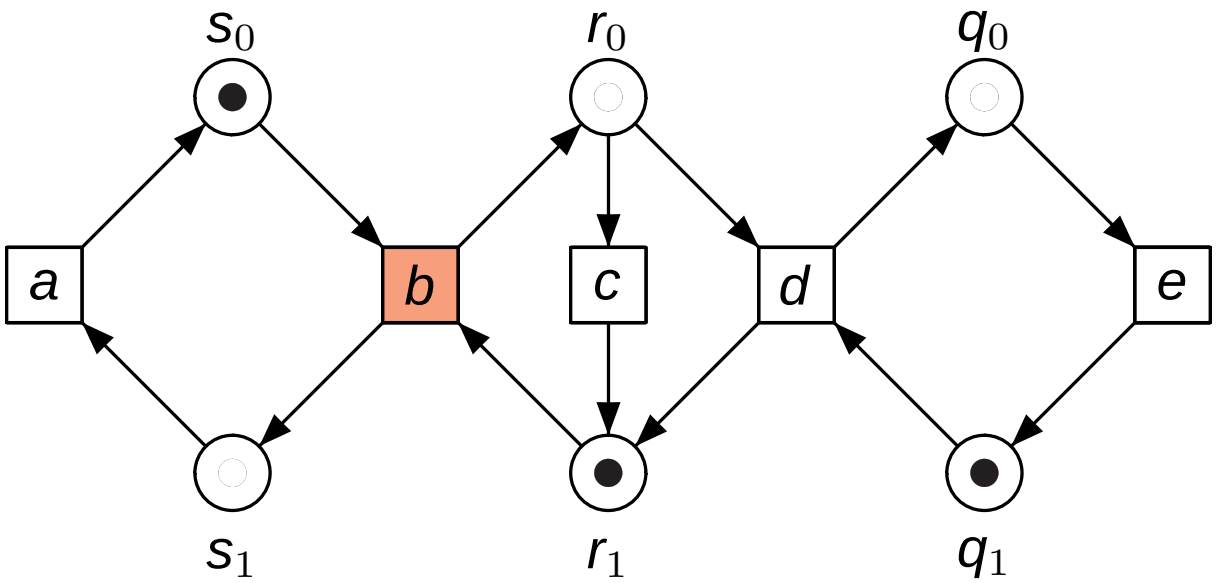
$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{b} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{e} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \xrightarrow{d} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{e} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$



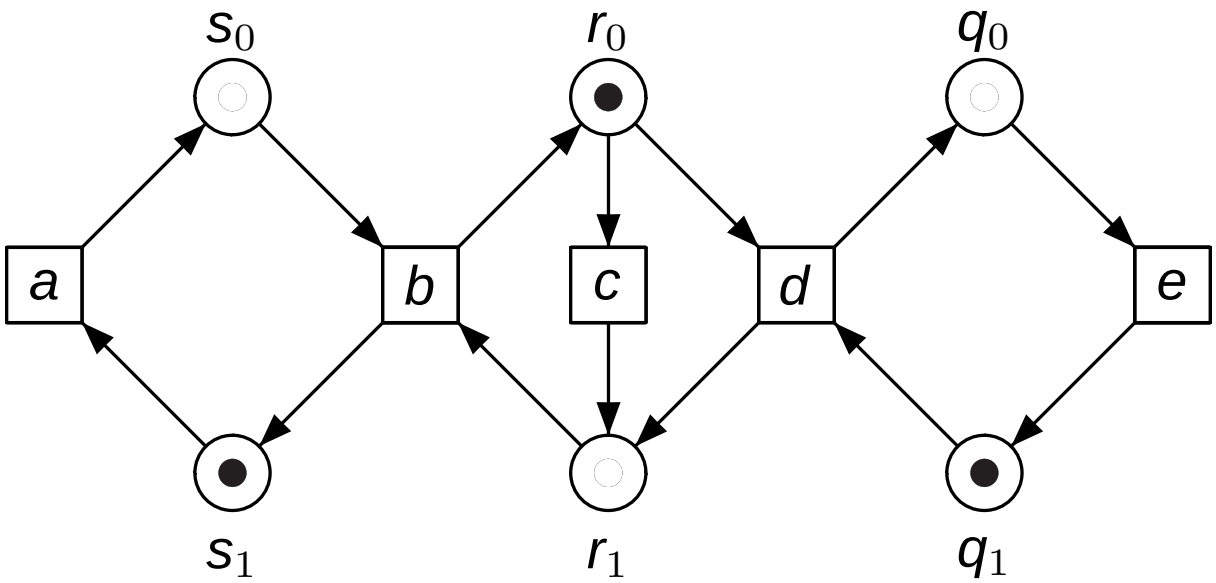
$$
\begin{matrix} s \\ r \\ q \end{matrix}
\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
\xrightarrow{c}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\xrightarrow{b}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\xrightarrow{e}
\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}
\xrightarrow{d}
\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}
\xrightarrow{e}
\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}
$$



$$
\begin{array}{c}
s \\ r \\ q
\end{array}
\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
\xrightarrow{c}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\xrightarrow{b}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\xrightarrow{e}
\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}
\xrightarrow{d}
\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}
\xrightarrow{e}
\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}
\xrightarrow{a}
\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}
$$

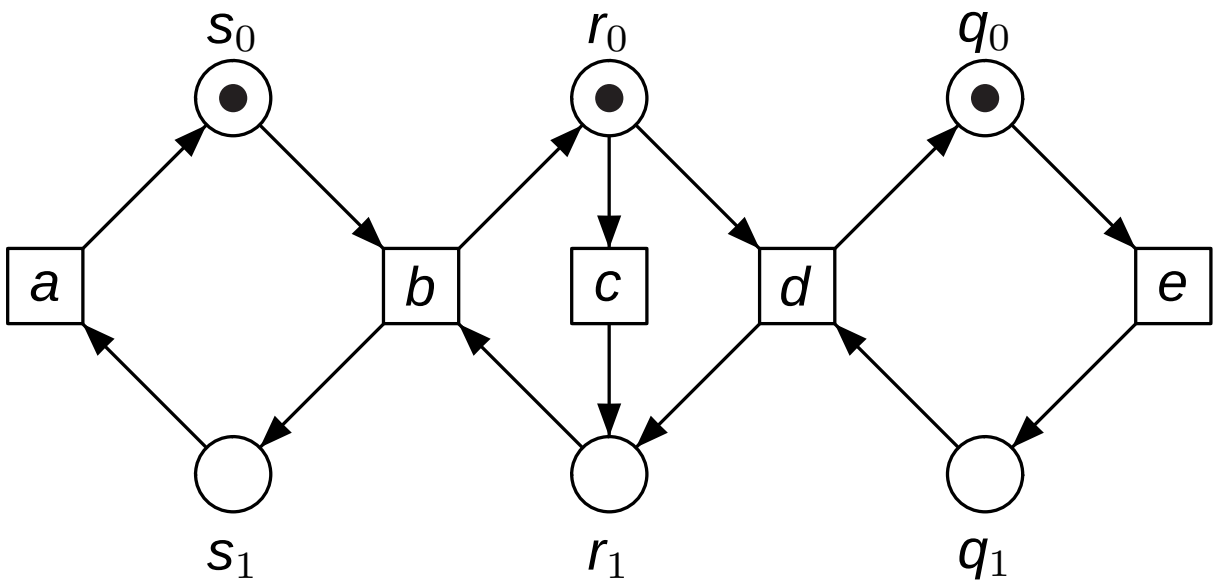


$$\begin{matrix} s \\ r \\ q \end{matrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{c} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{b} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{e} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \xrightarrow{d} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{e} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \xrightarrow{a} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

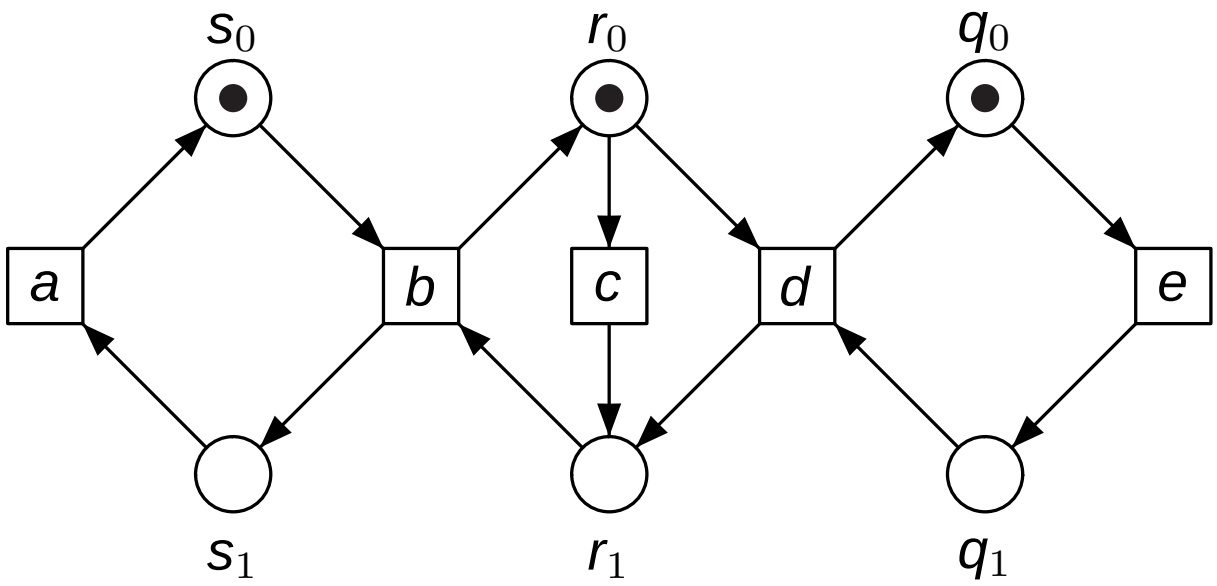


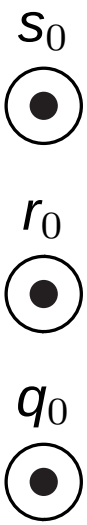
$$
\begin{array}{c}
s \\ r \\ q
\end{array}
\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
\xrightarrow{c}
\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}
\xrightarrow{b}
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\xrightarrow{e}
\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}
\xrightarrow{d}
\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}
\xrightarrow{e}
\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}
\xrightarrow{a}
\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}
\xrightarrow{b}
\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}
$$

True-concurrency semantics: concurrent executions

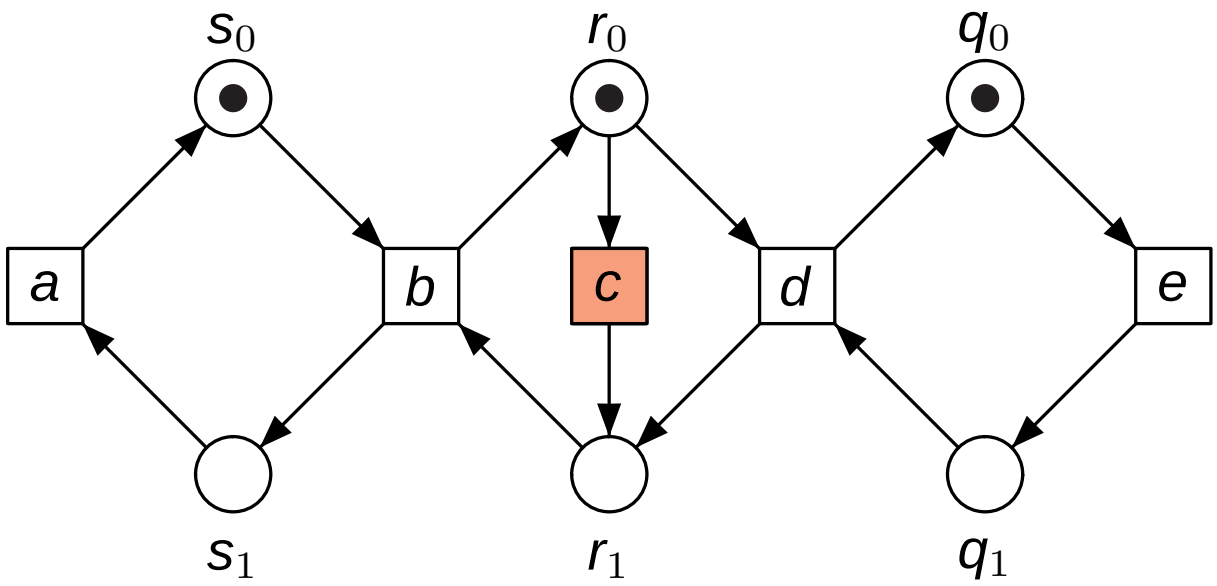


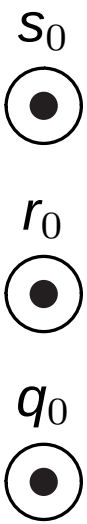
True-concurrency semantics: concurrent executions



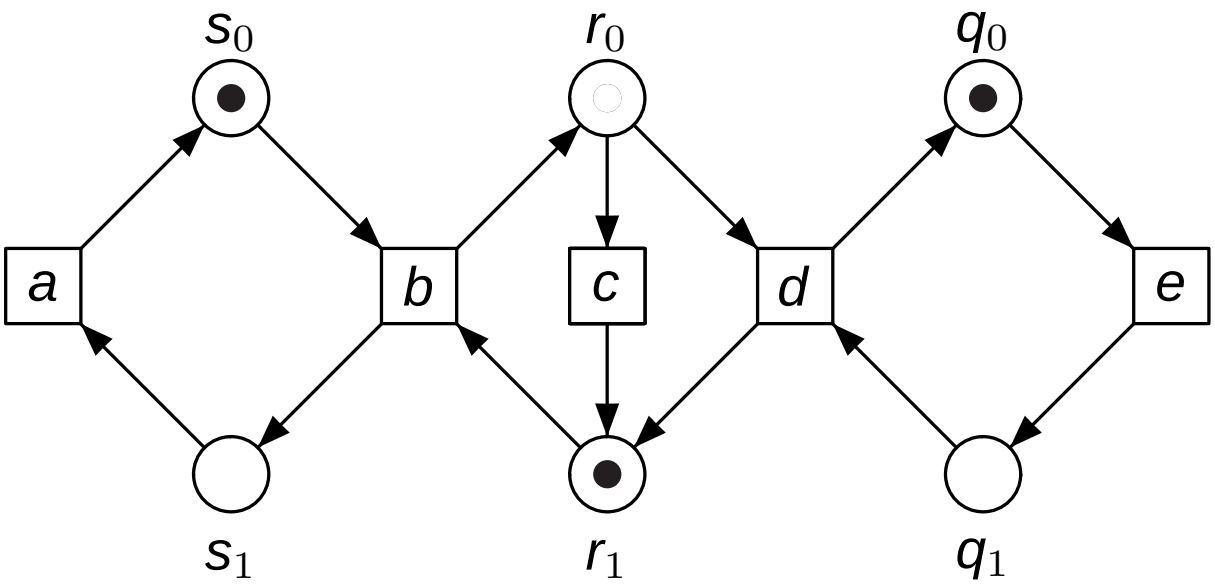


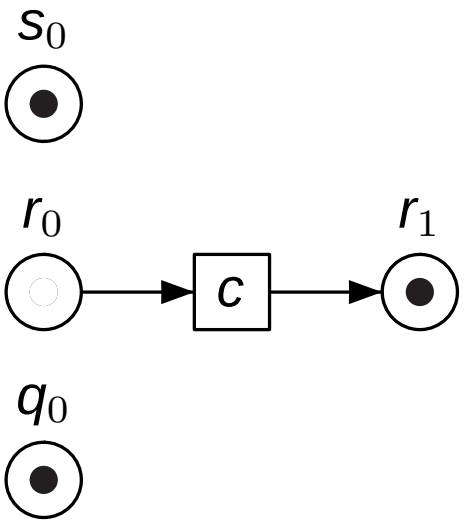
True-concurrency semantics: concurrent executions



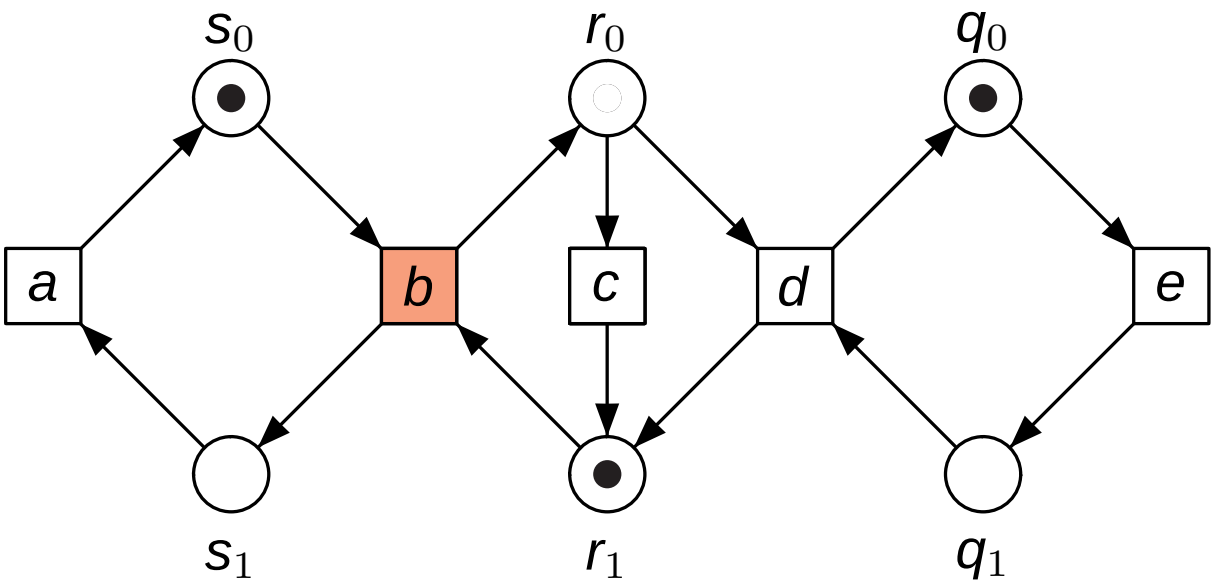


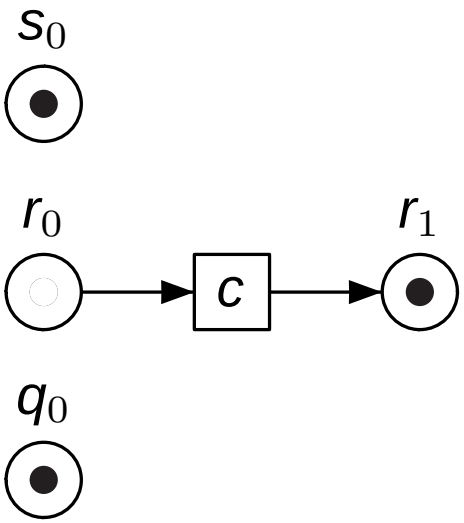
True-concurrency semantics: concurrent executions



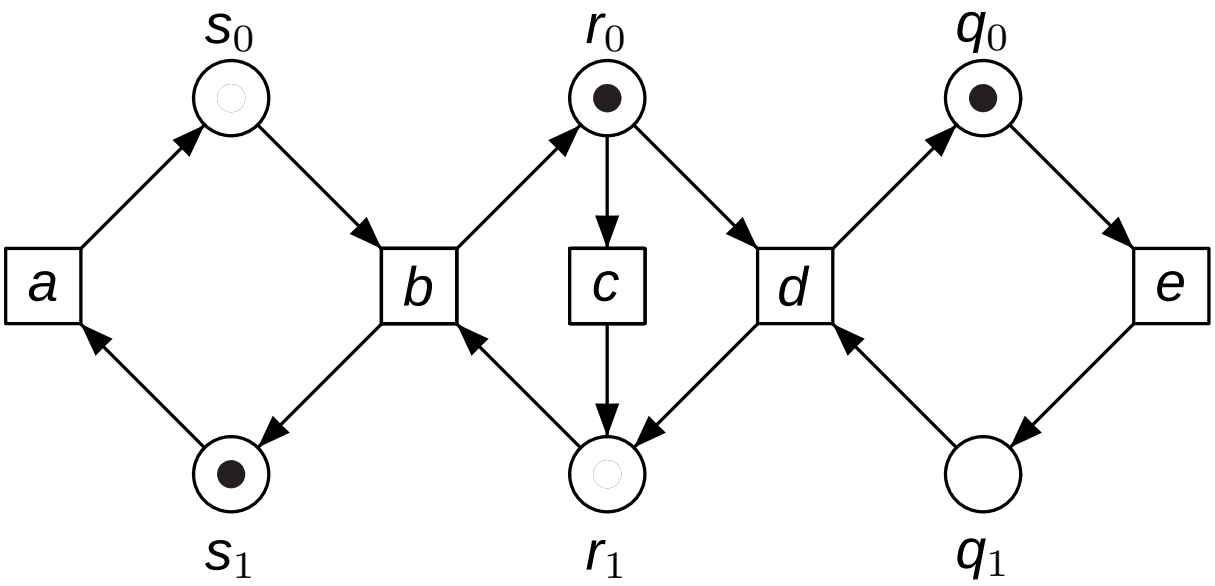


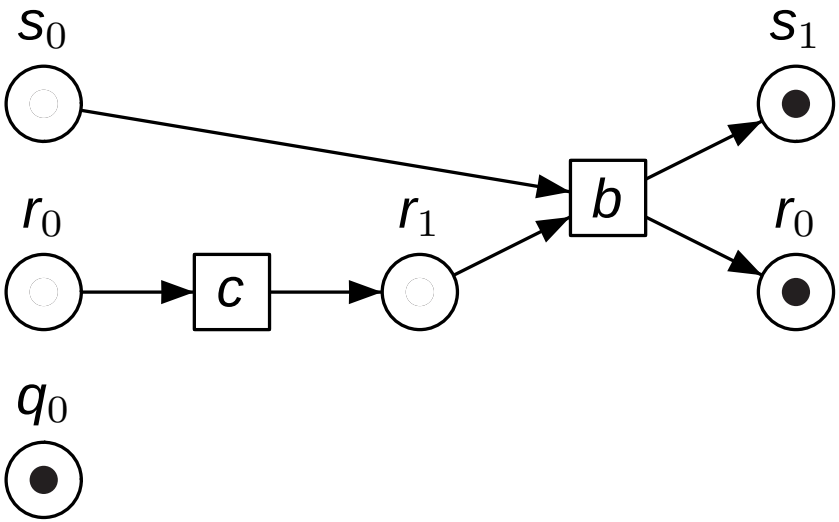
True-concurrency semantics: concurrent executions



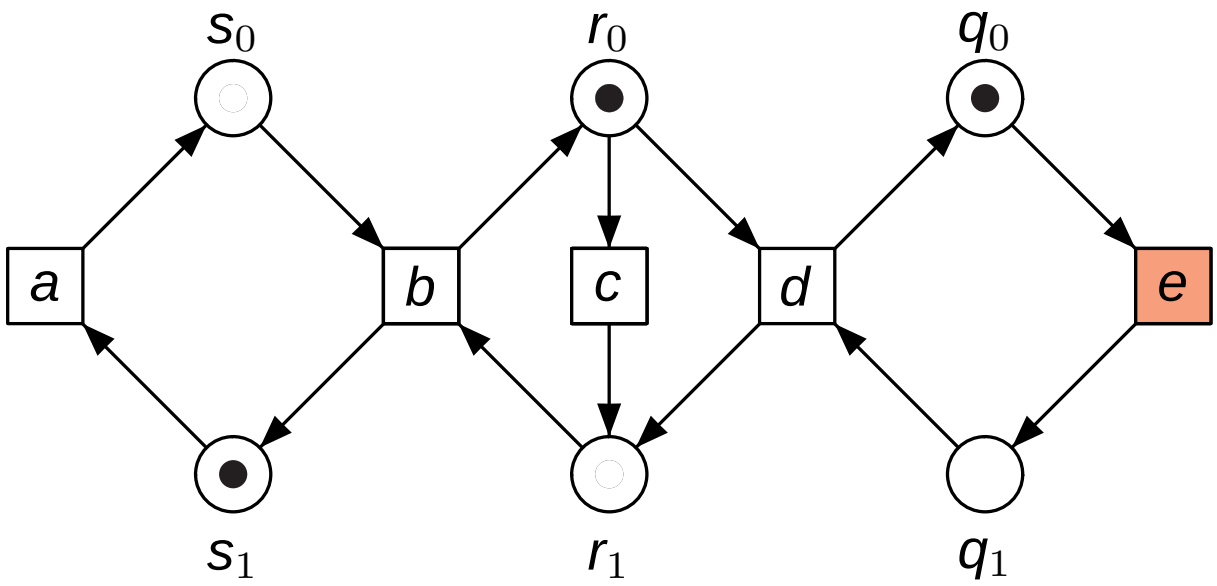


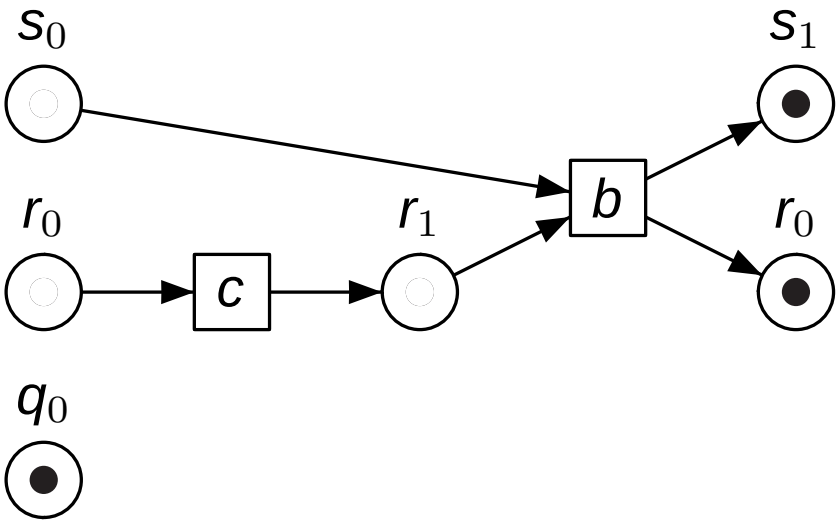
True-concurrency semantics: concurrent executions



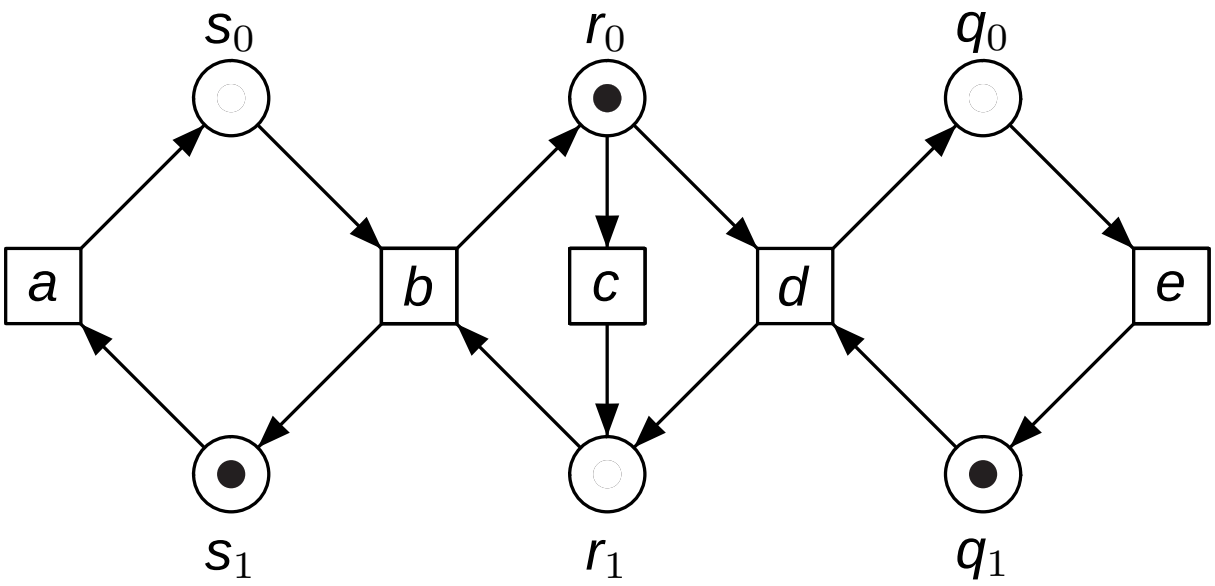


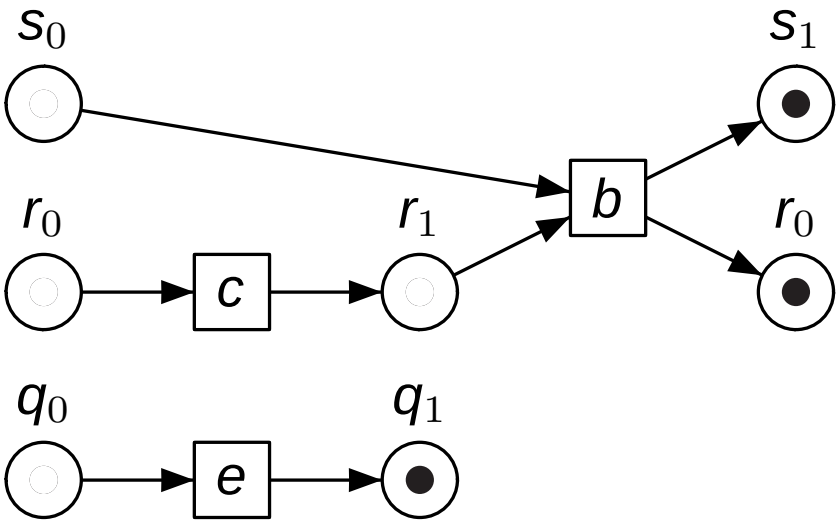
True-concurrency semantics: concurrent executions



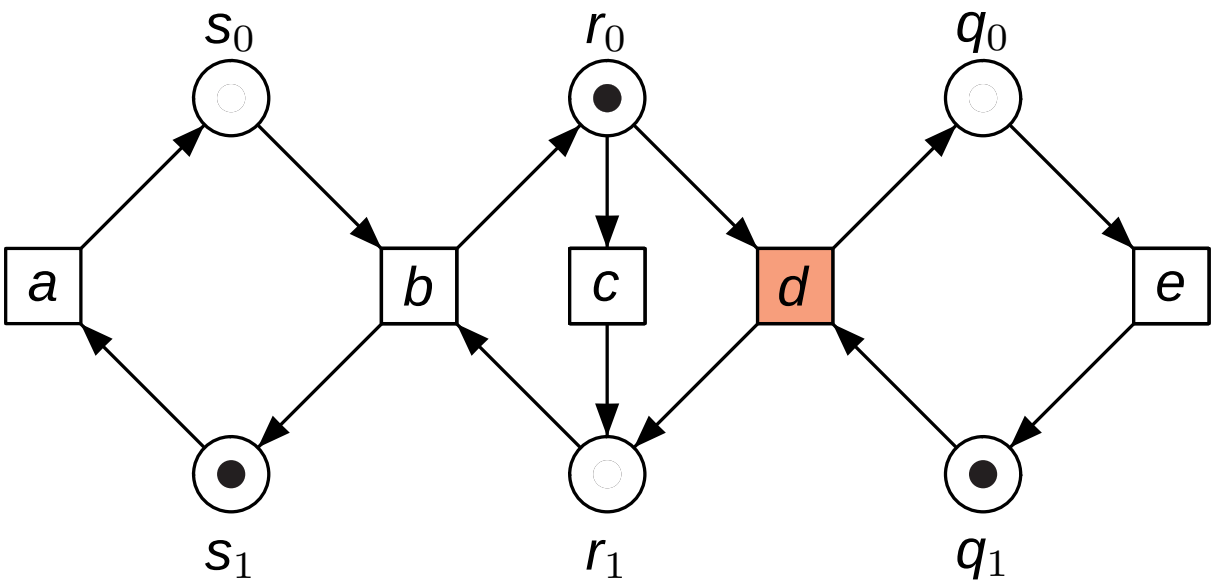


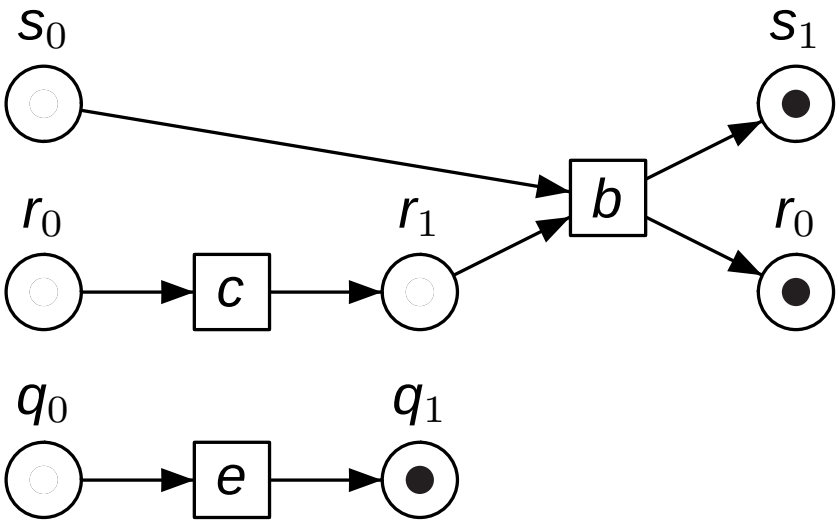
True-concurrency semantics: concurrent executions



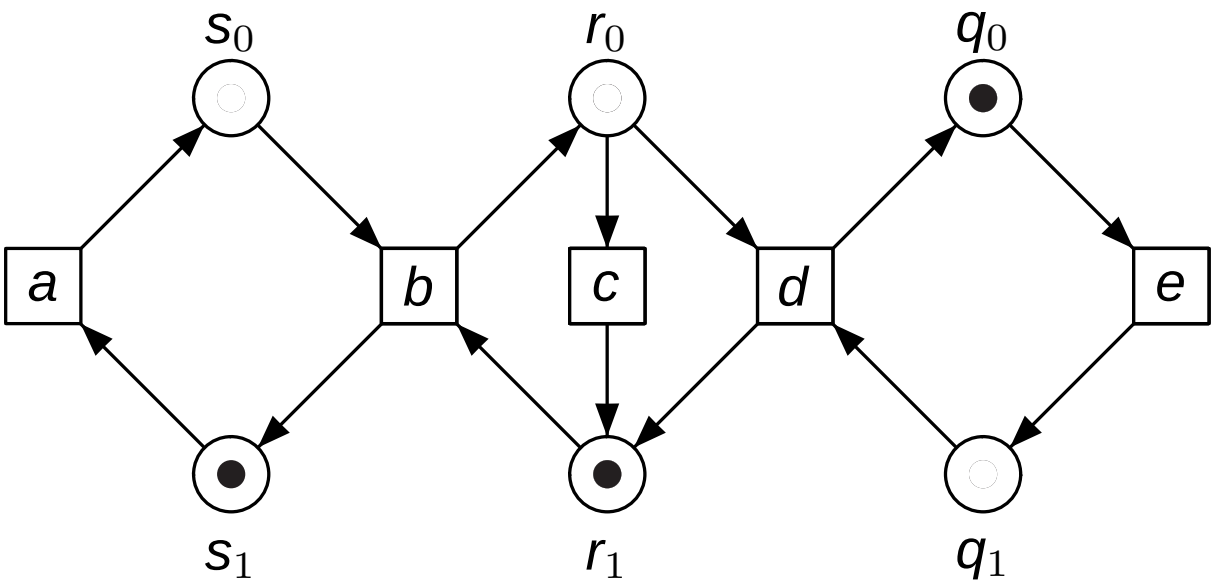


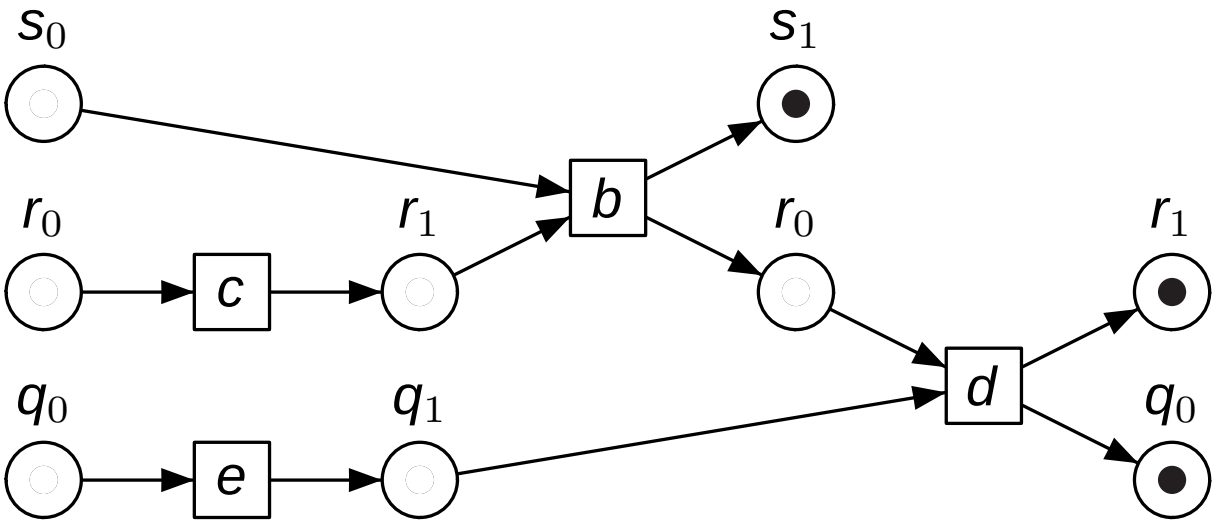
True-concurrency semantics: concurrent executions



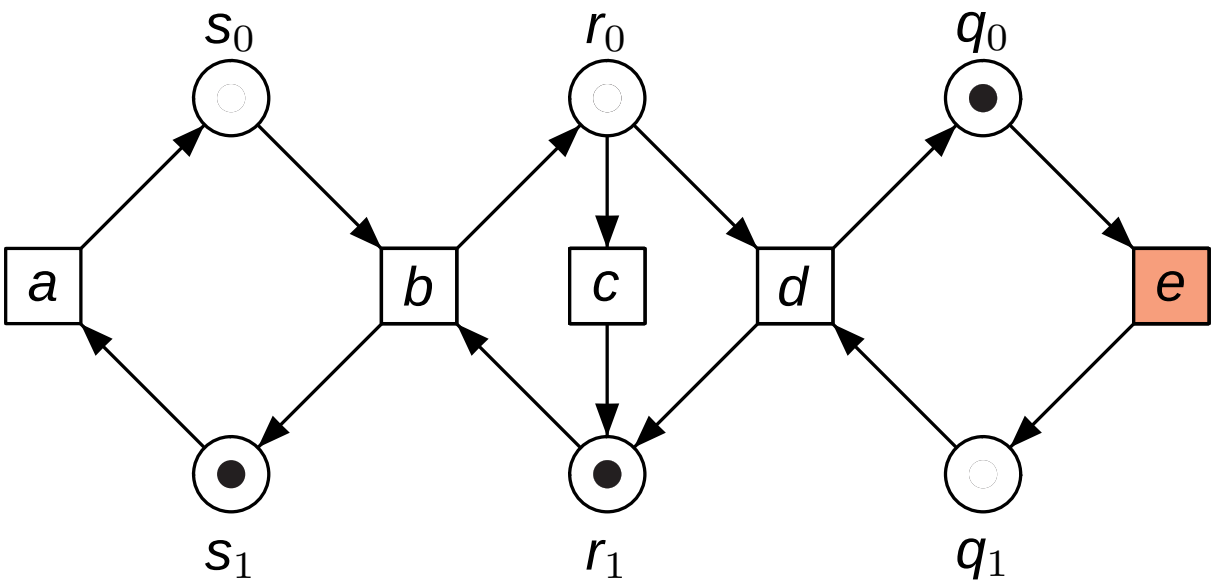


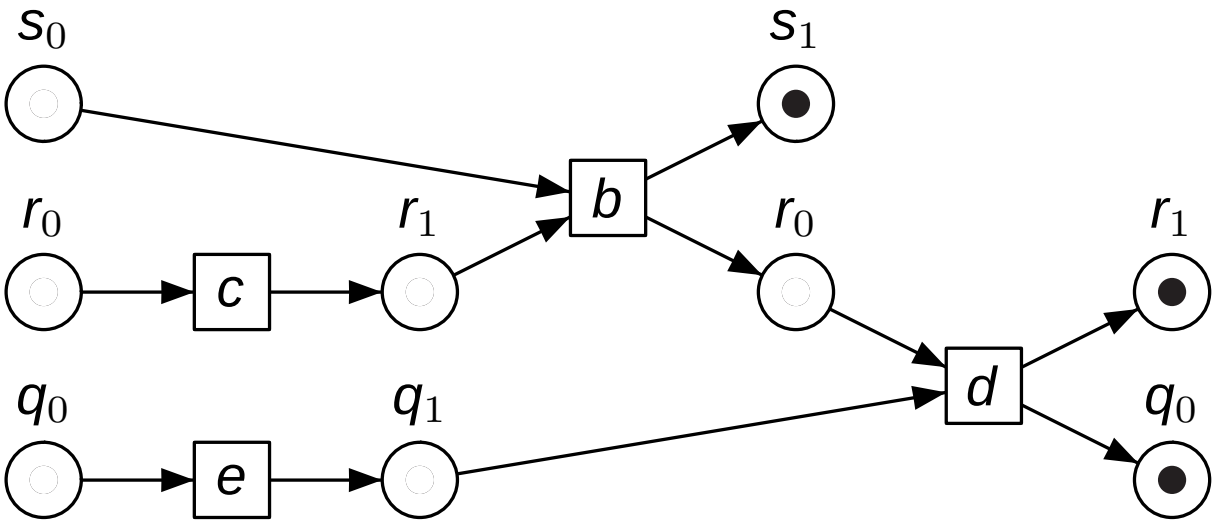
True-concurrency semantics: concurrent executions



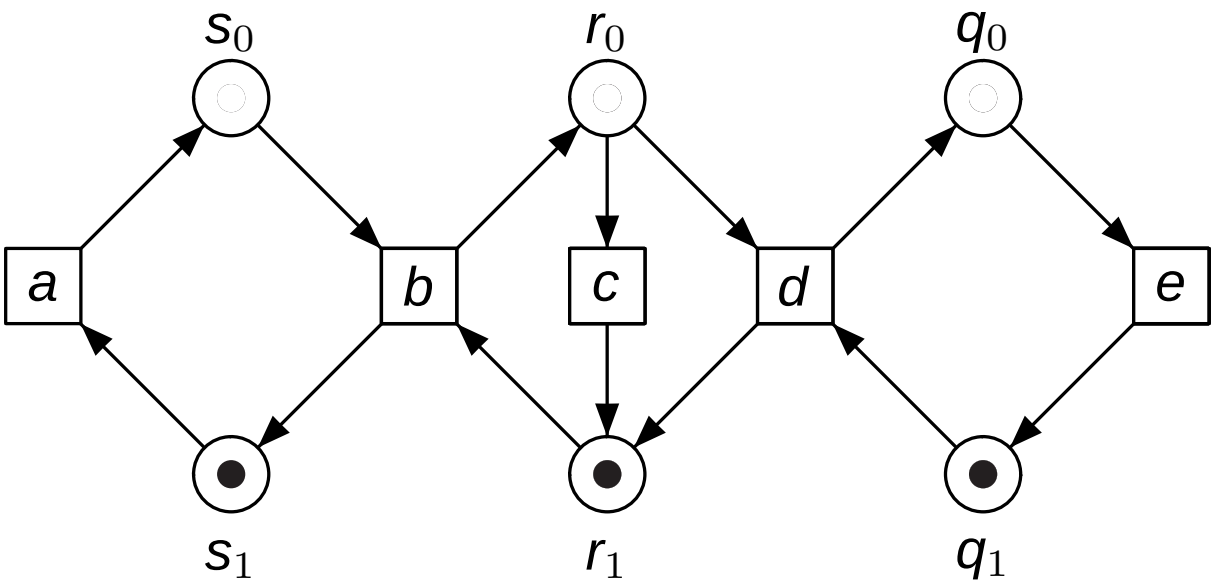


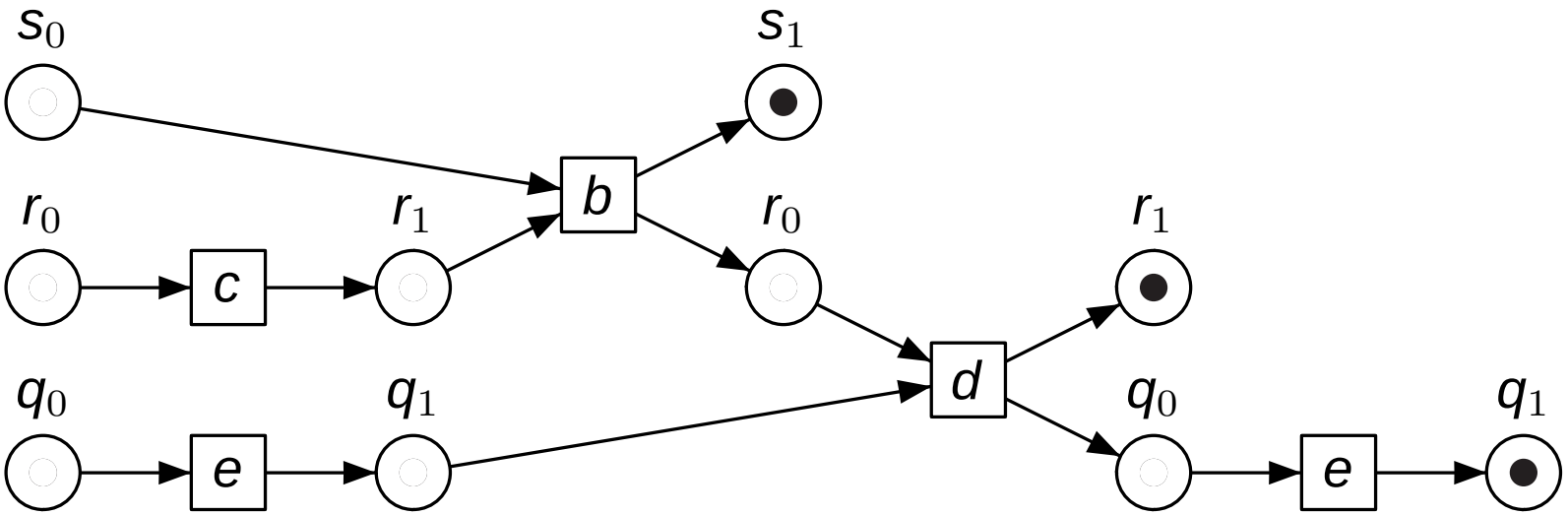
True-concurrency semantics: concurrent executions



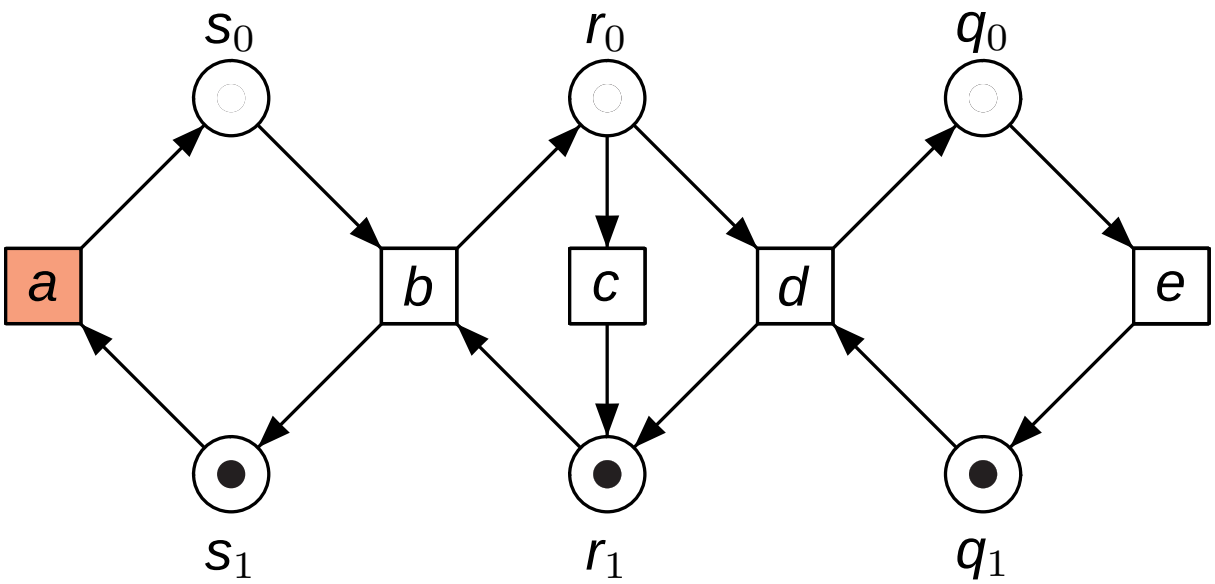


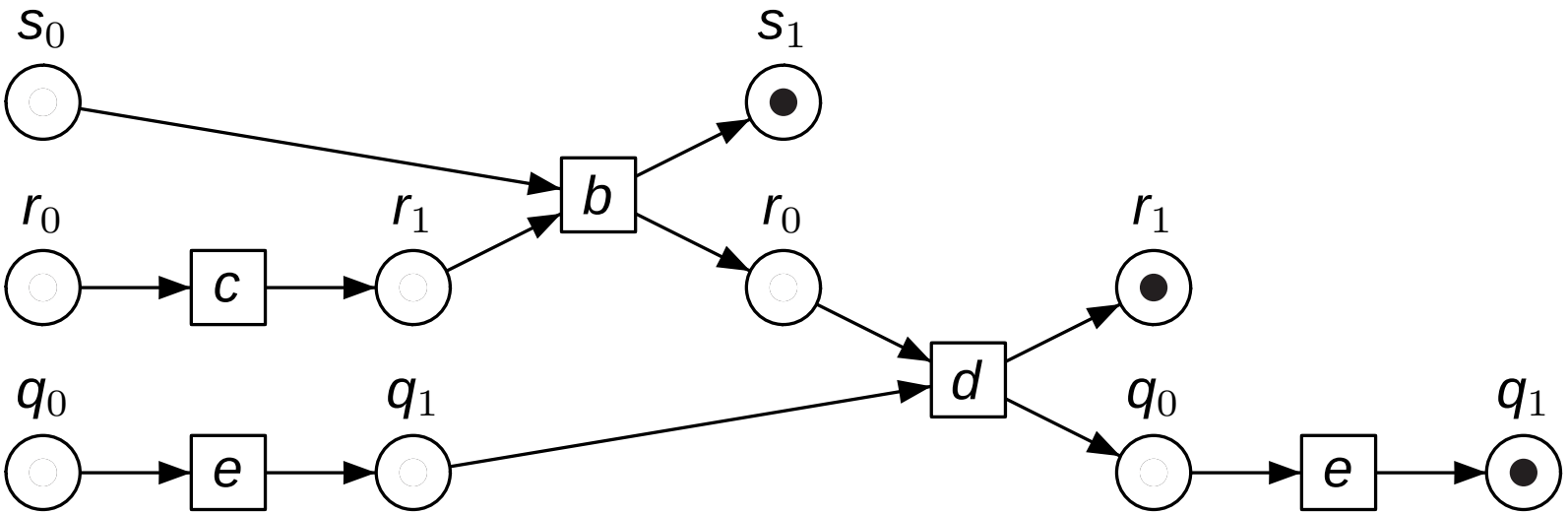
True-concurrency semantics: concurrent executions



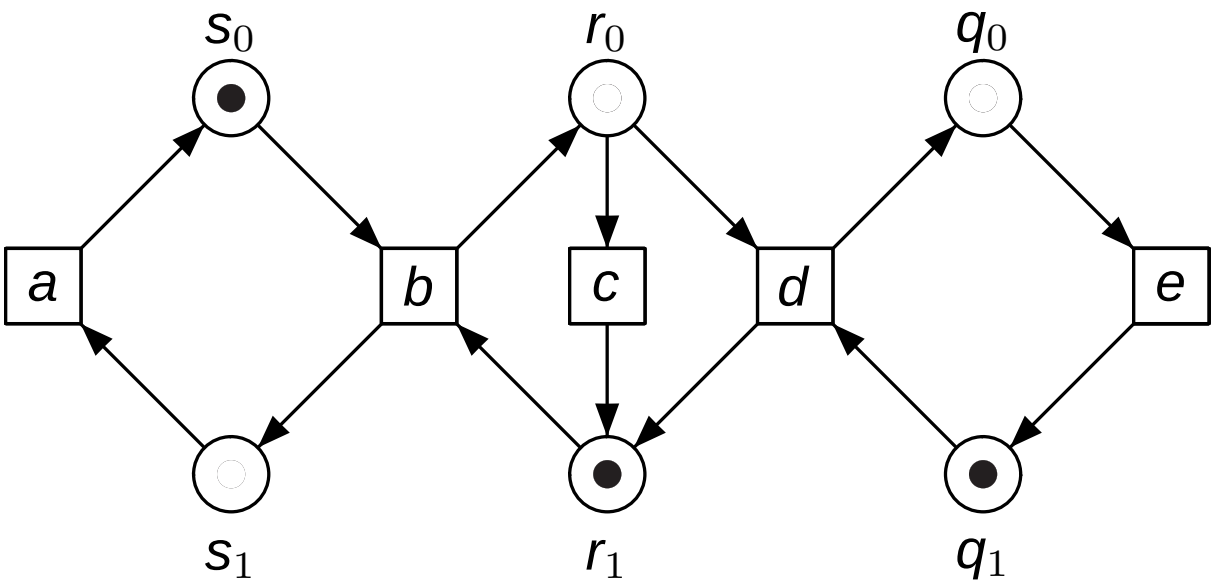


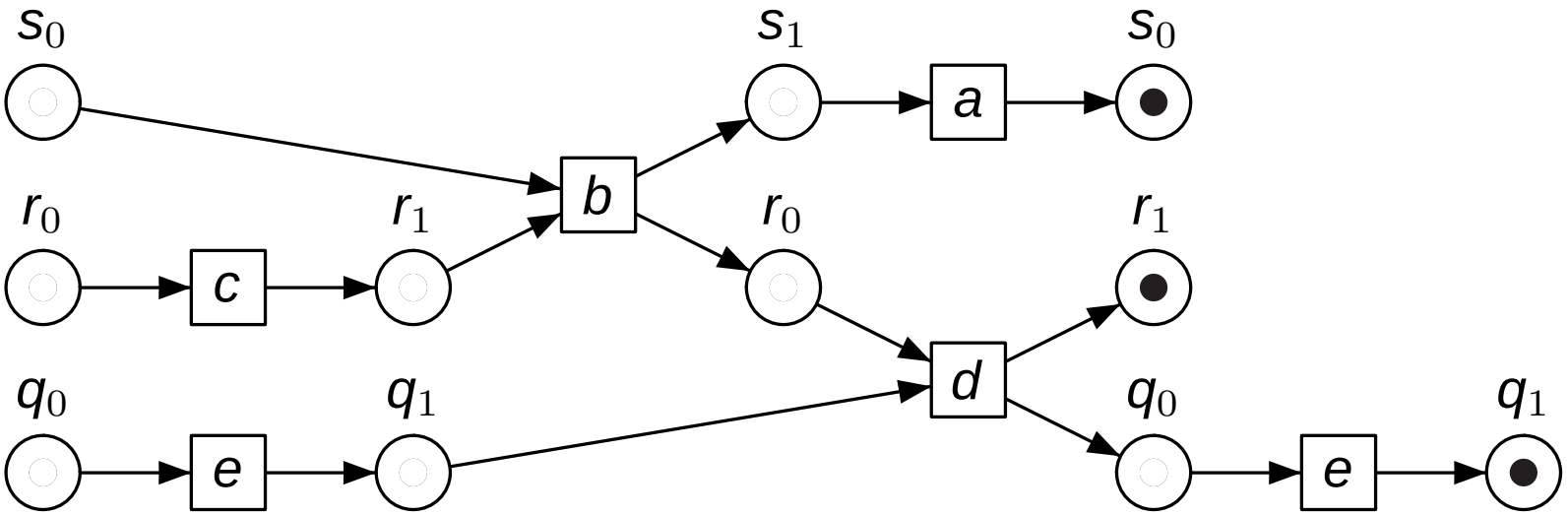
True-concurrency semantics: concurrent executions



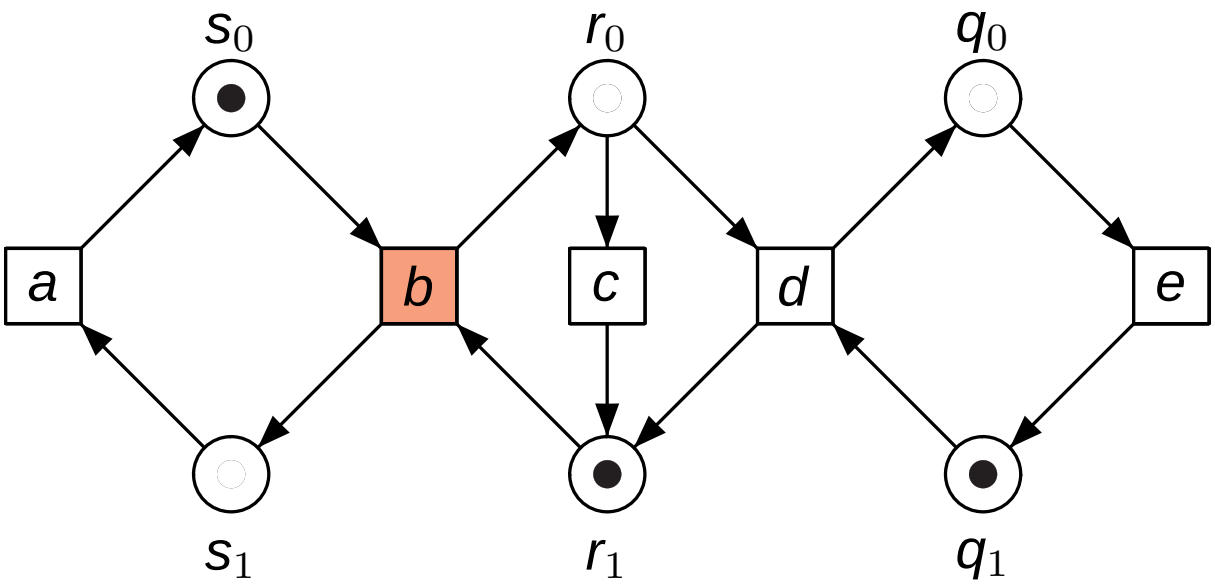


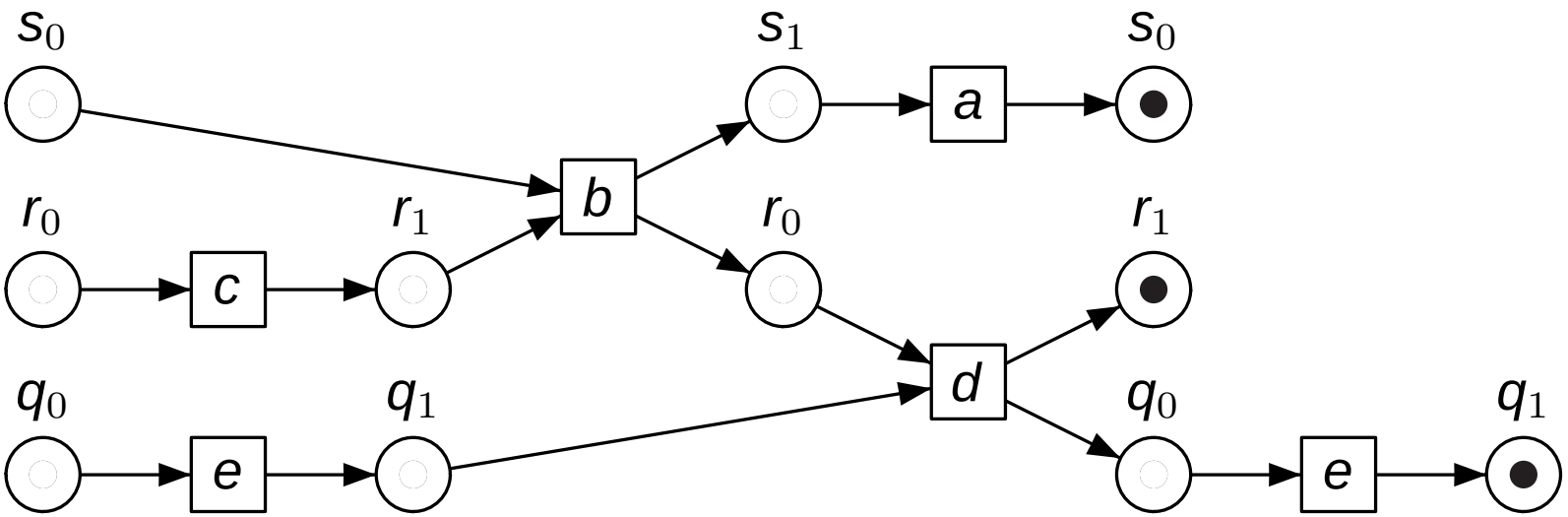
True-concurrency semantics: concurrent executions



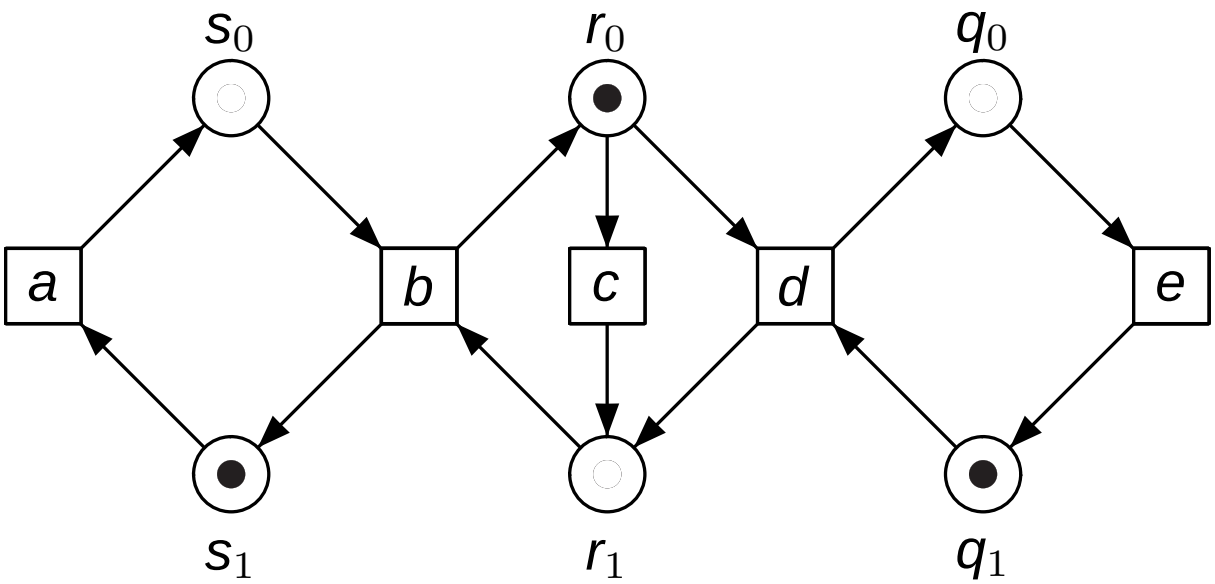


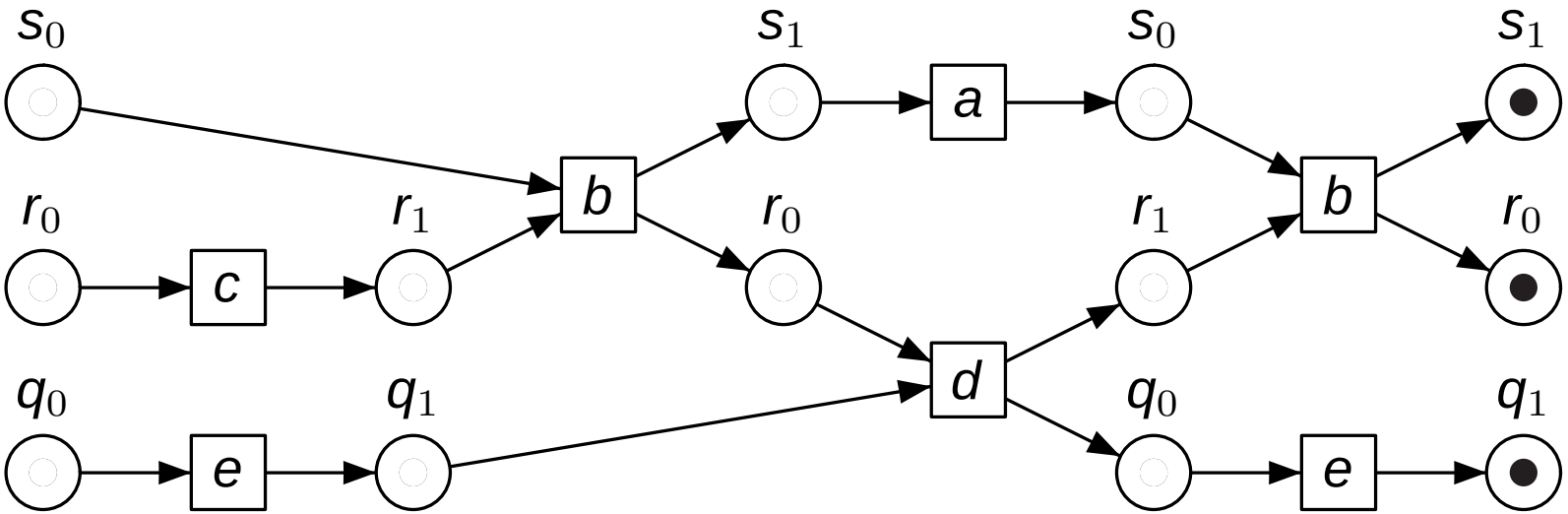
True-concurrency semantics: concurrent executions





True-concurrency semantics: concurrent executions





Executions vs. concurrent executions

The argument for executions:

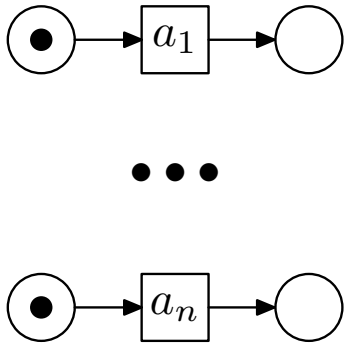
> The global clock assumption is adequate for practical purposes, and leads to nice mathematics

The argument against executions:

> The global clock assumption does not correspond to physical reality, and leads to awkward representations of simple phenomena

The standard example

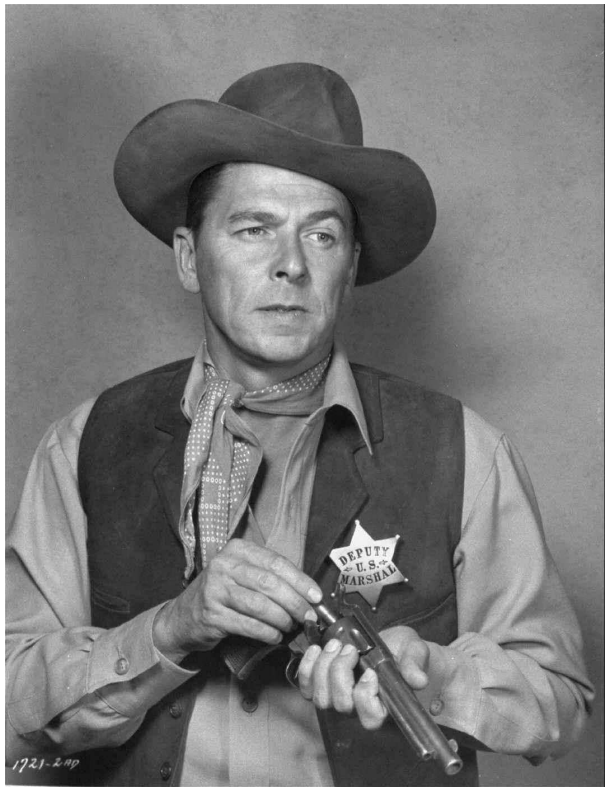
A system composed of n independent components



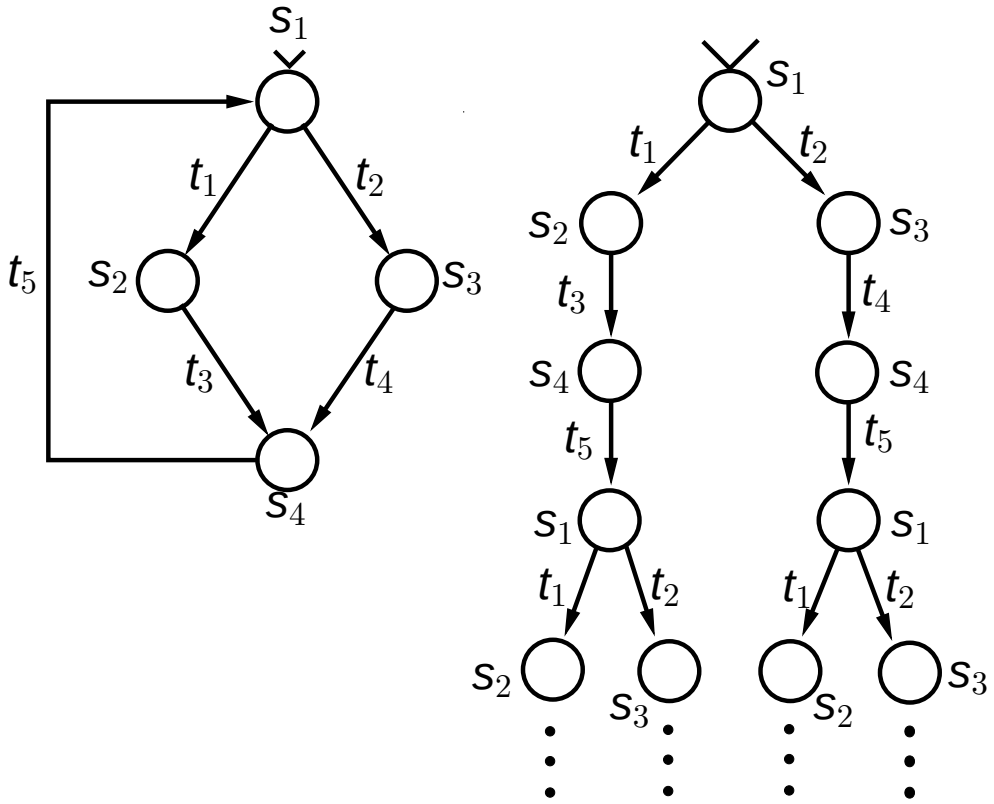
has $n!$ different executions, and 2 global states.

The system has only one concurrent execution of size $O(n)$.

1981

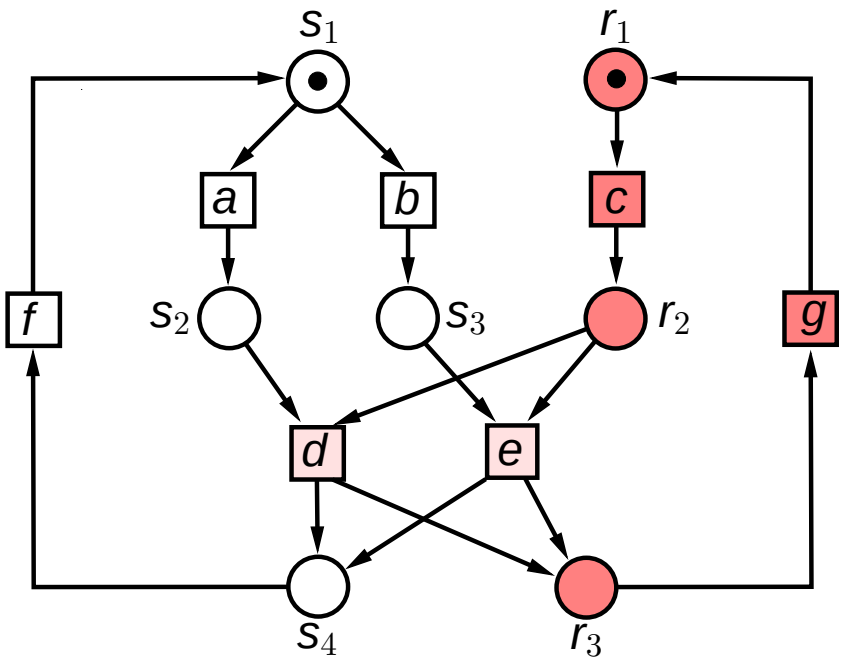


Execution trees

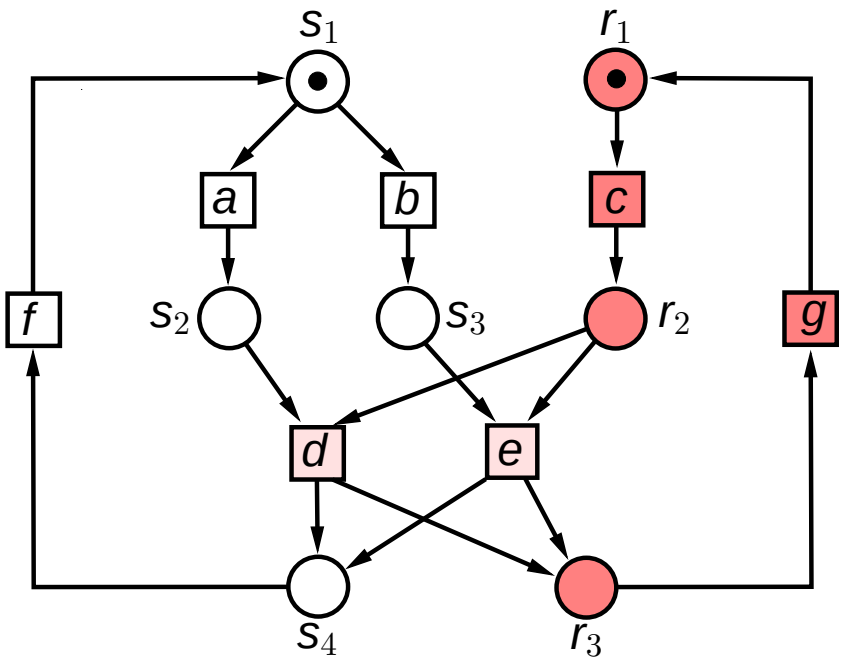


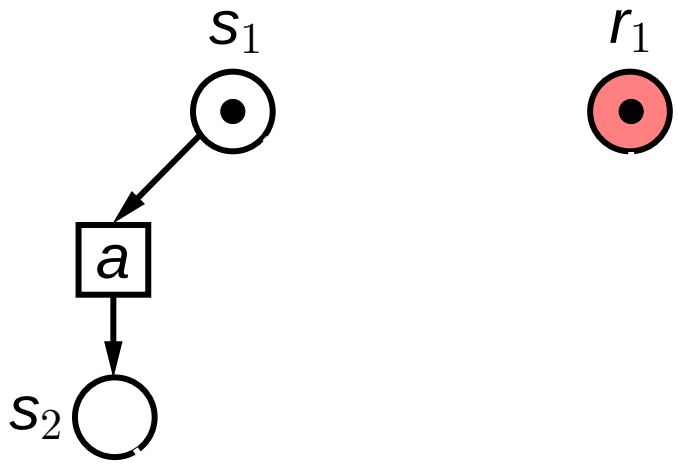
Nielsen, Plotkin, Winskel '81:
What could be a "concurrent execution tree" ?

Unfolding a Petri net

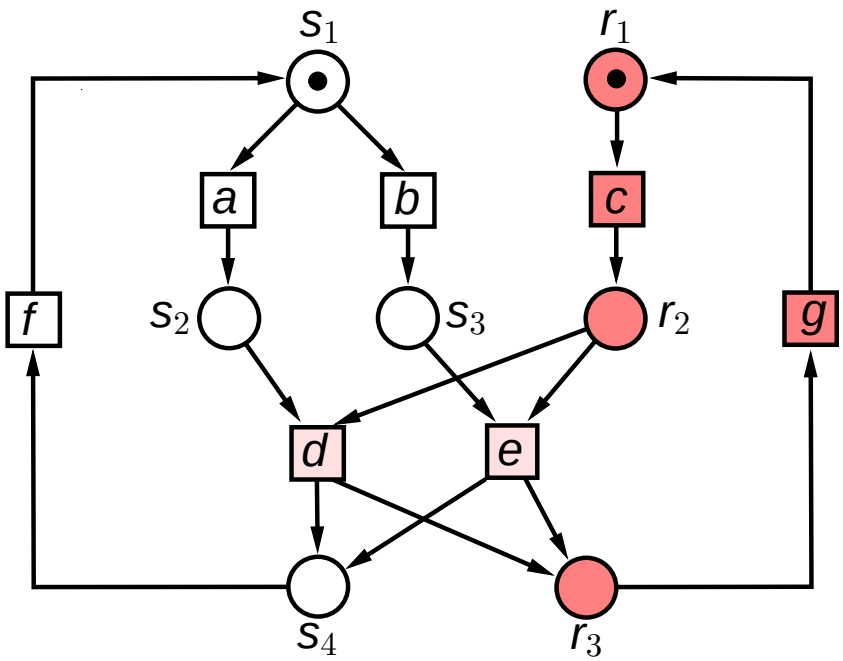


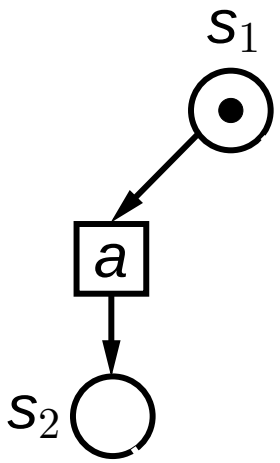
Unfolding a Petri net

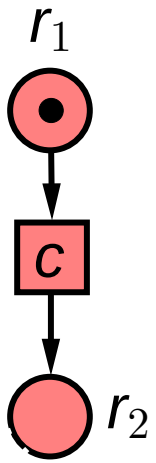




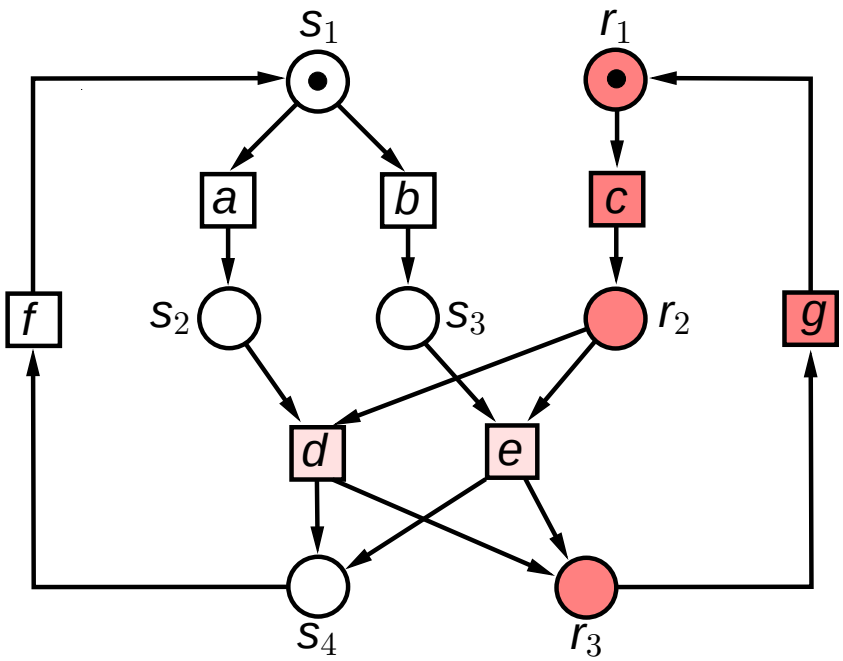
Unfolding a Petri net

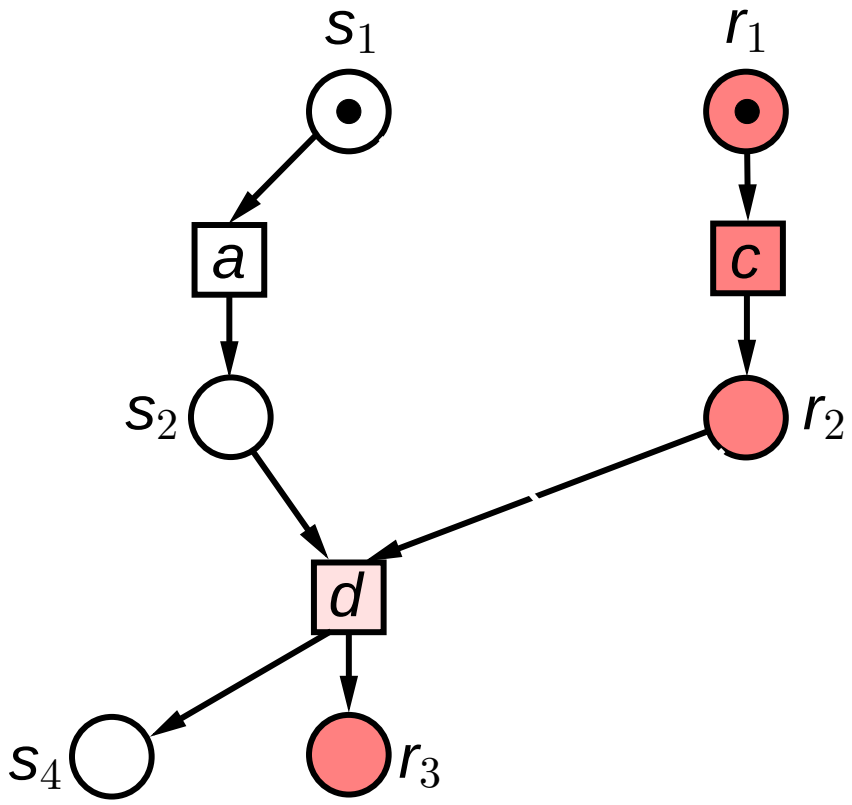




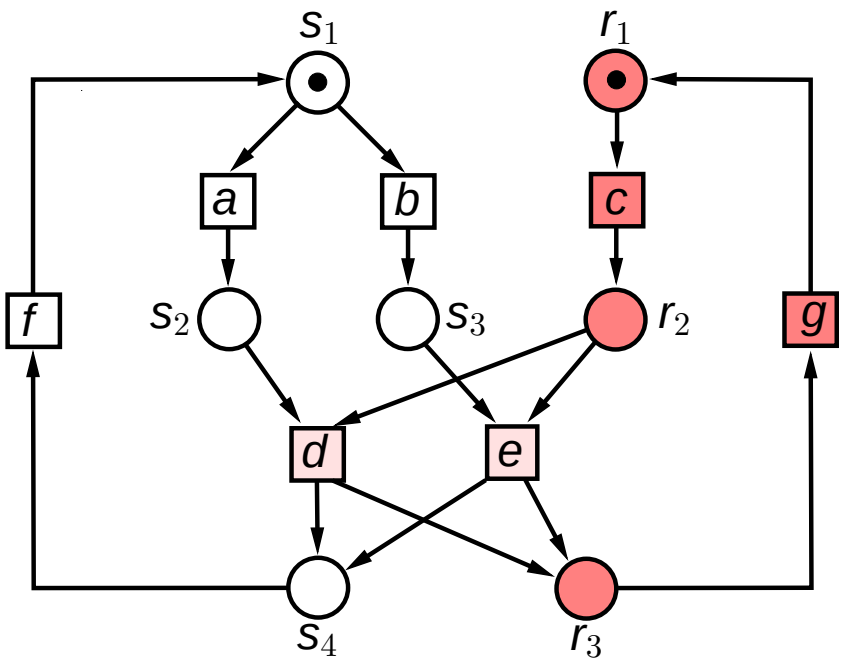


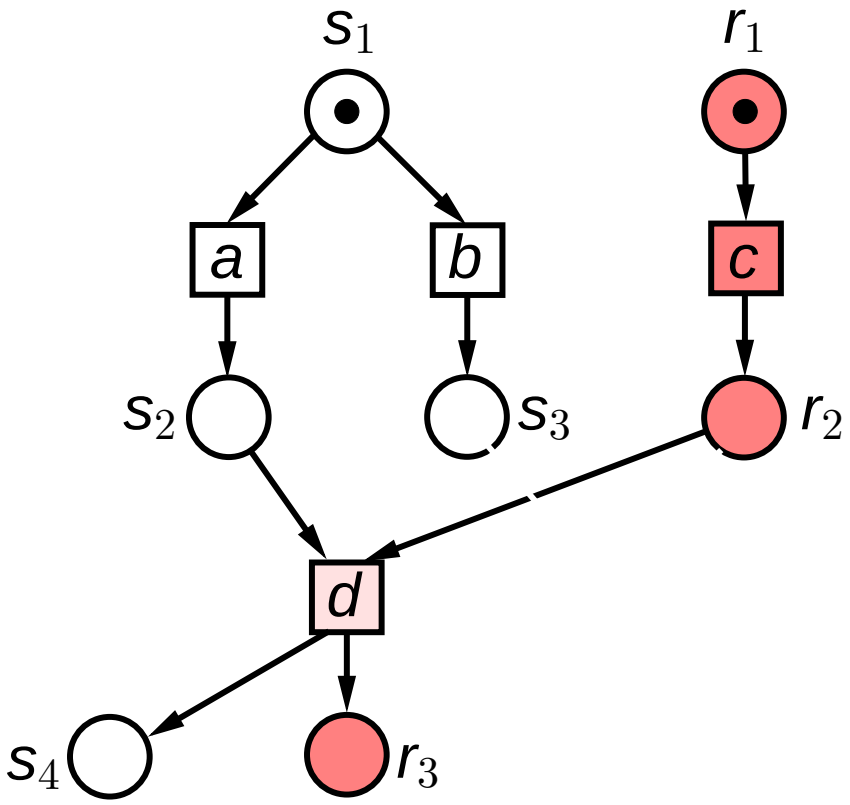
Unfolding a Petri net



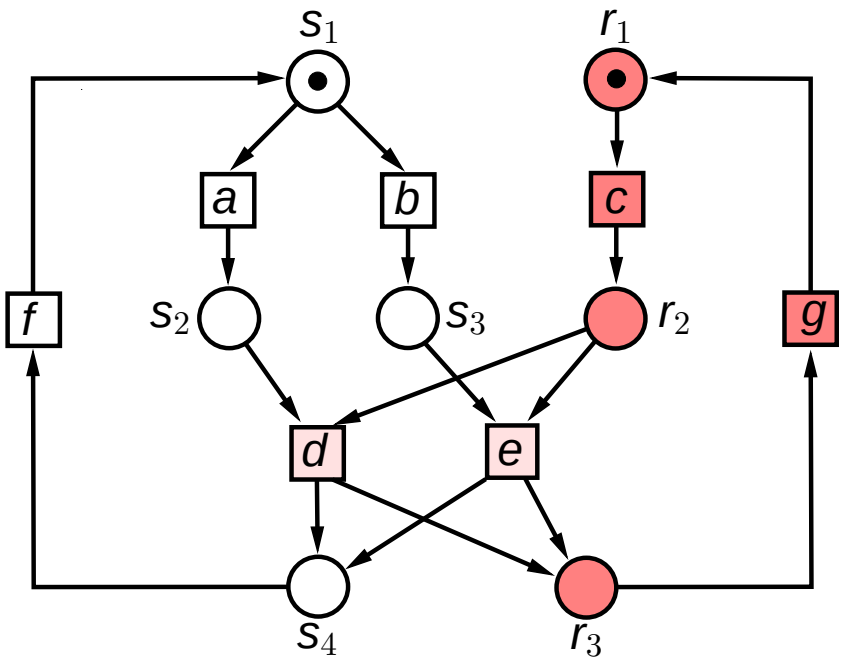


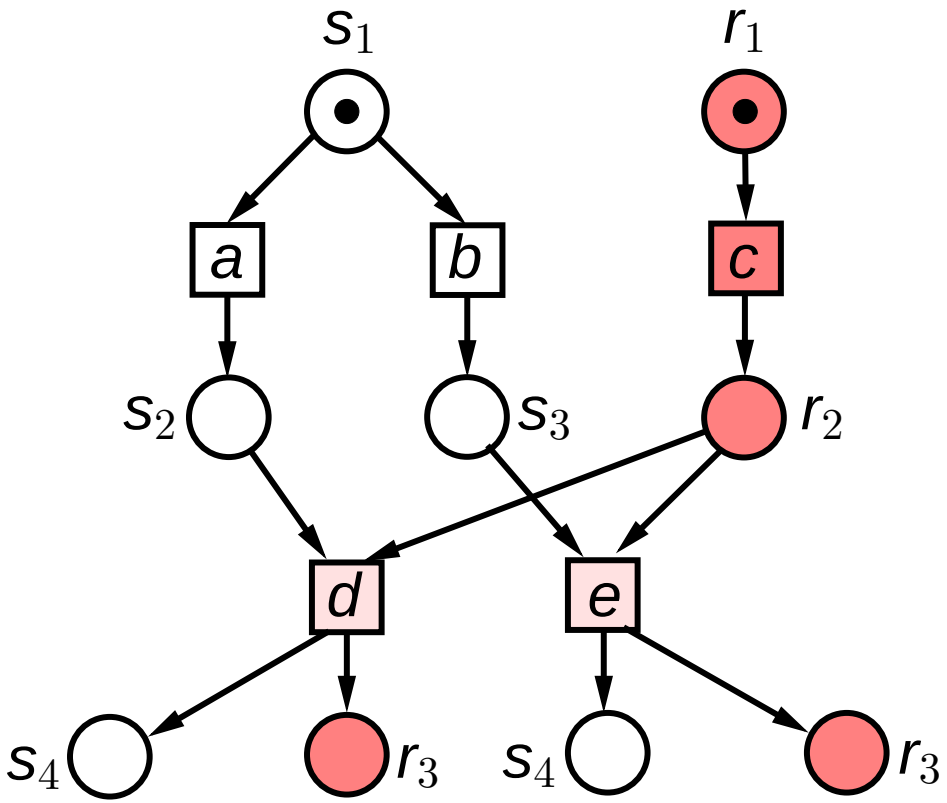
Unfolding a Petri net



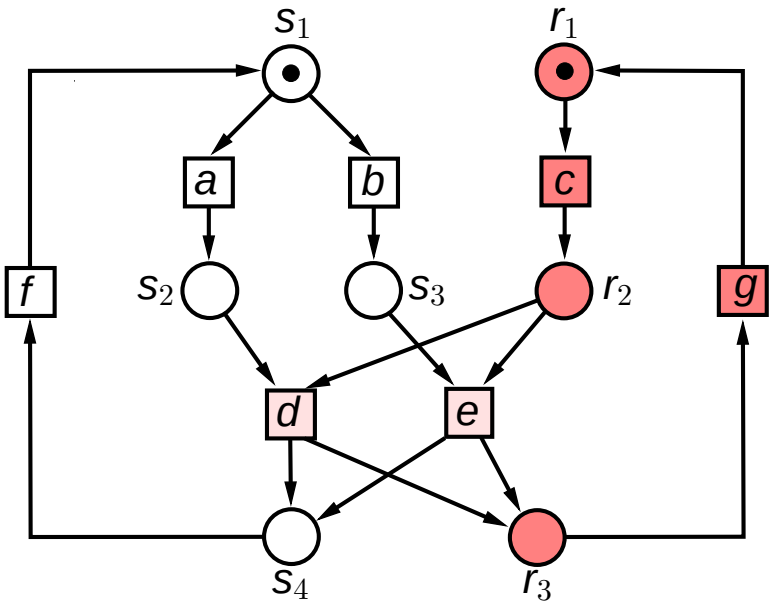


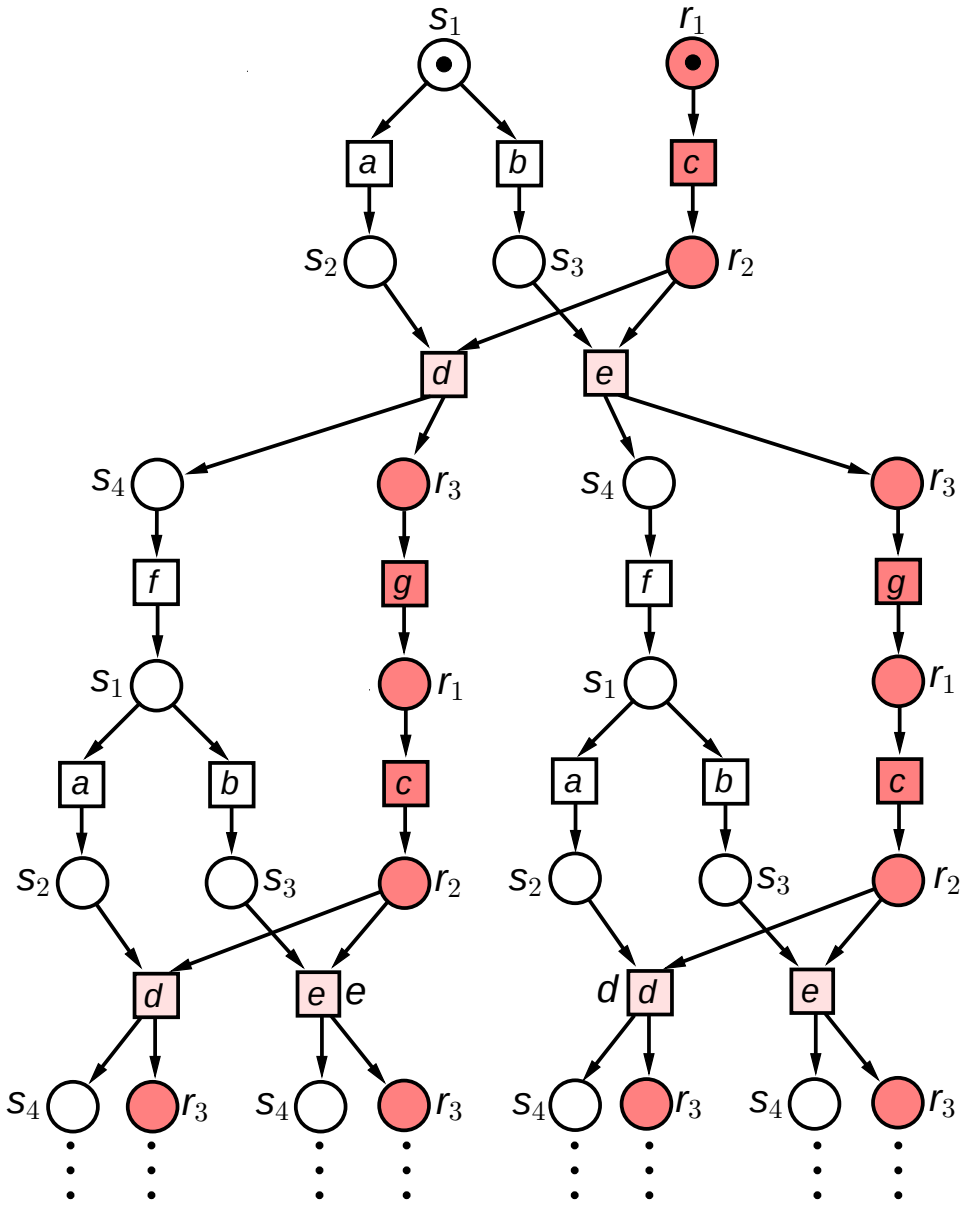
Unfolding a Petri net





The unfolding





Motivation of Nielsen, Plotkin and Winskel purely semantic

Denotational semantics of the concurrent behaviour of a Petri net

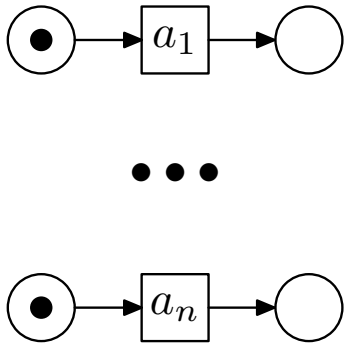
Definition of a domain analogous to Scott's domain for computable functions

Extension of Scott's thesis 'he computable functions are the continuous functions" to concurrency.

1992

The state-explosion problem

A system composed of n independent components



has 2^n reachable states.

Its unfolding is the system itself, and has size $O(n)$

McMillan: can the unfolding help palliate the state-explosion problem?

The executability problem

Executability: does some execution of the net contain a given action ?

Fact: model-checking safety properties $\longrightarrow$ executability problem.

Goal: design search algorithms that instead of exploring the execution tree explore the unfolding.

Hope: the algorithms need to explore a small prefix of the unfolding, instead of a large prefix of the execution tree.

Search algorithm

A search algorithm is determined by:

- A search strategy.
  Fixes which event to add next.
  In this talk we assume deterministic strategies.

- A termination condition.
  Fixes which leaves of the current prefix are terminals, i.e., nodes whose successors need not be explored.

Searching the execution tree

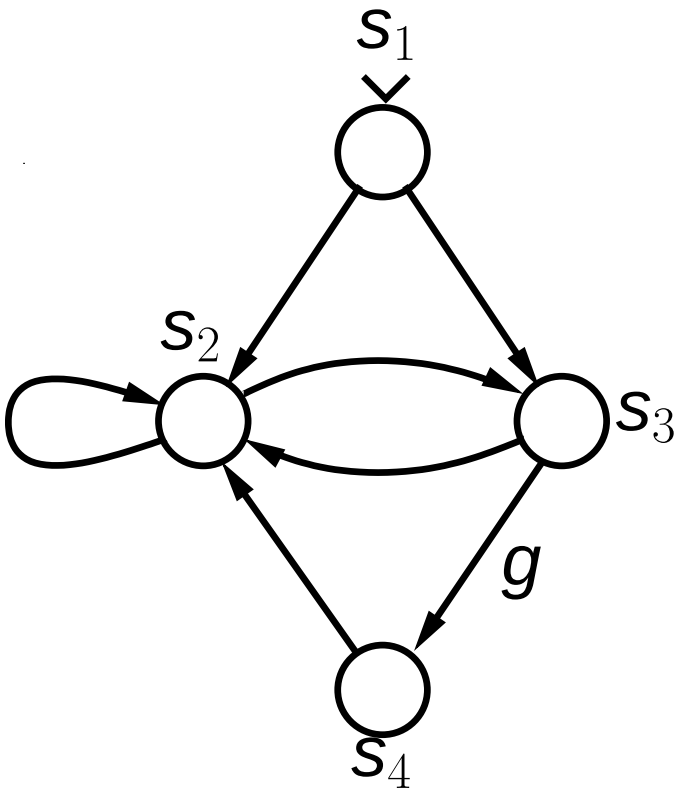
Search algorithm for executability of an action g

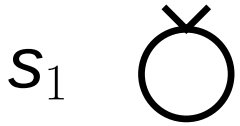
Search strategy: any.

Termination condition: An event is a terminal if

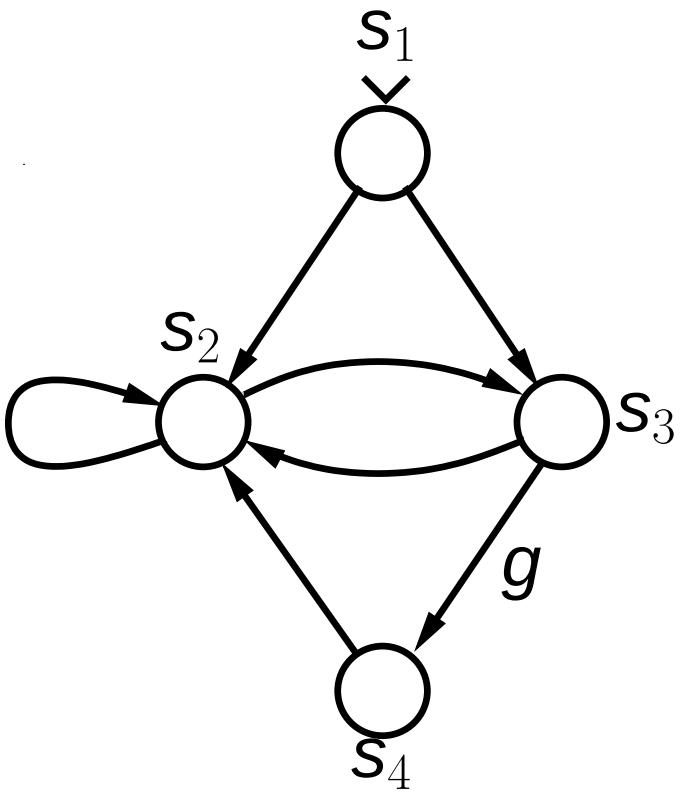
- it is labeled by g or,
- it leads to the same state as another already explored event.

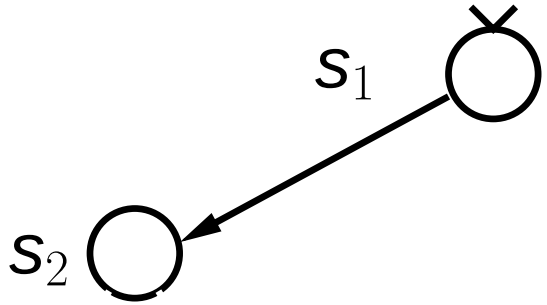
Example



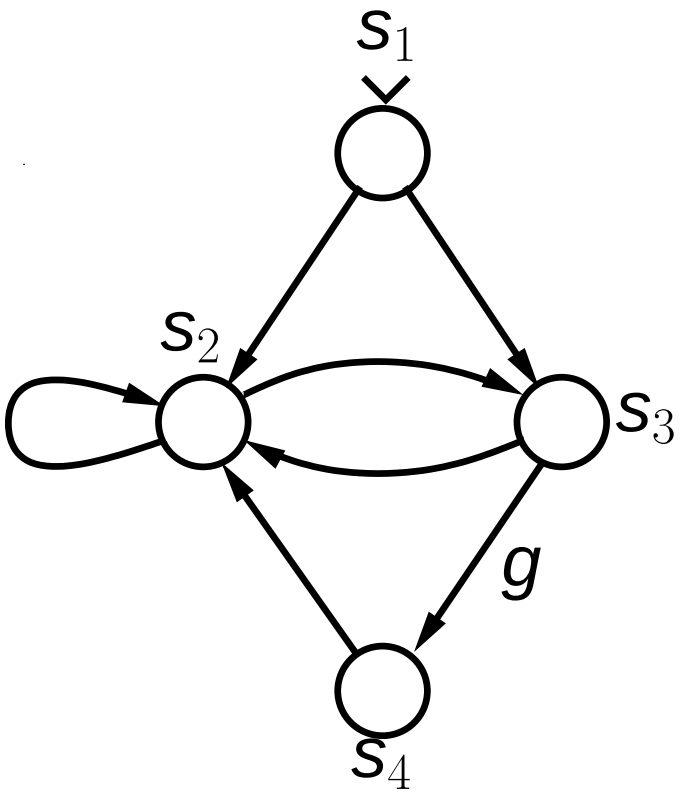


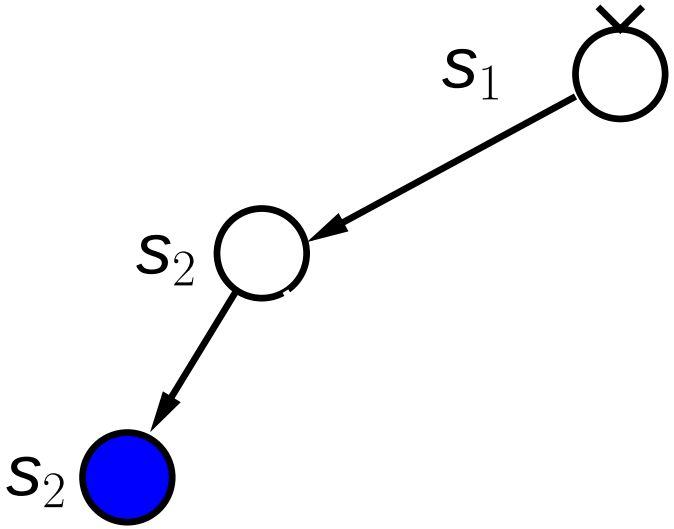
Example



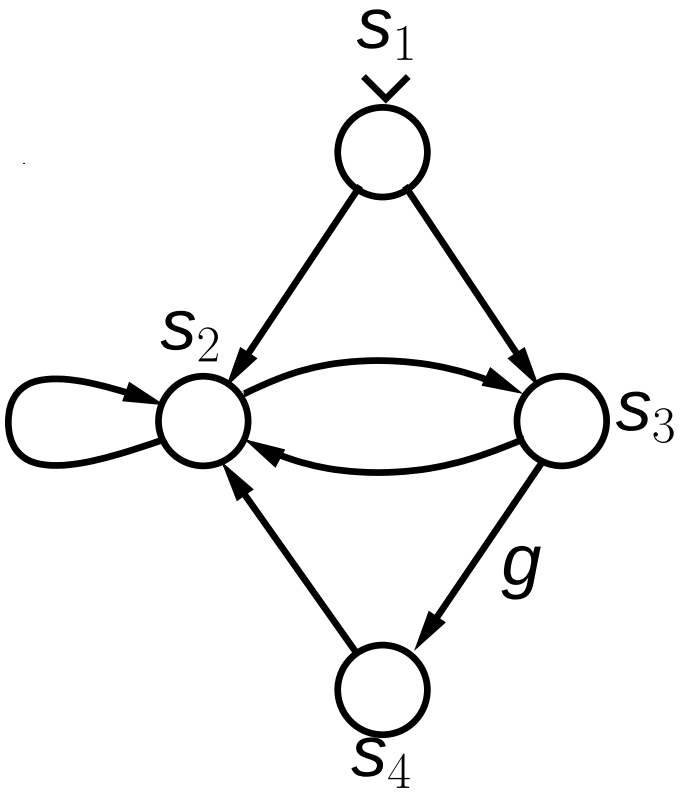


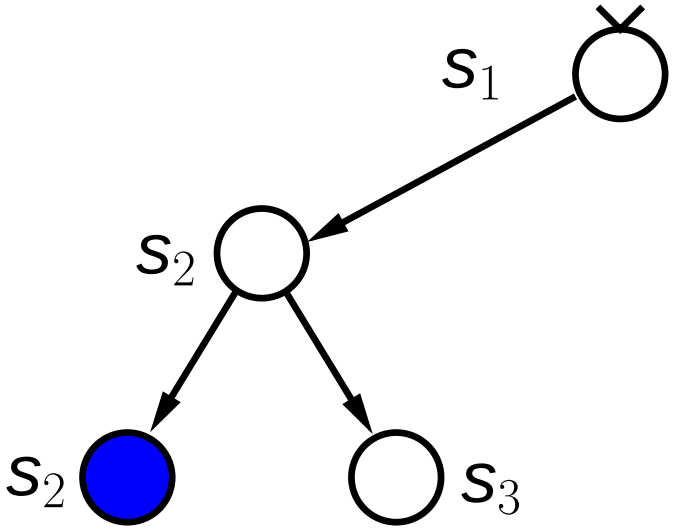
Example



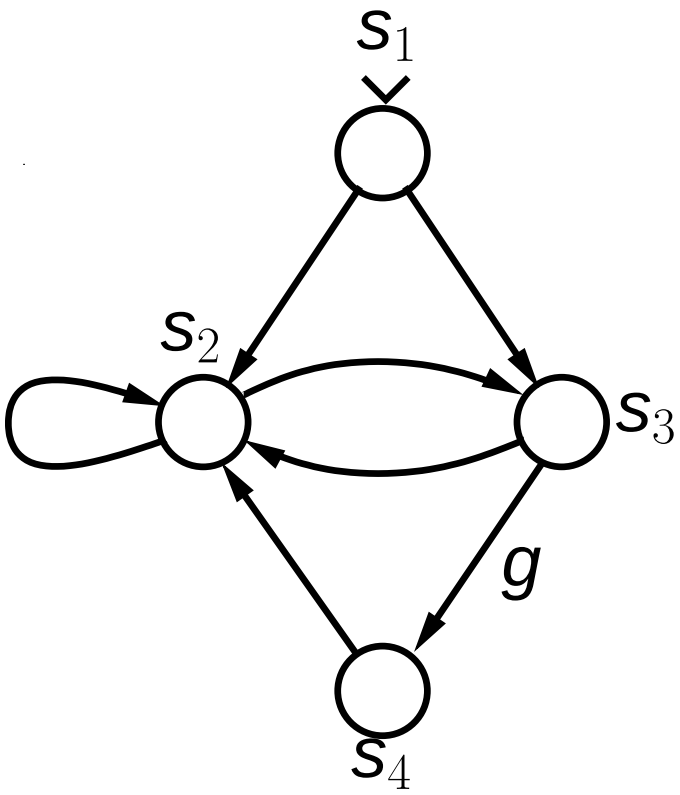


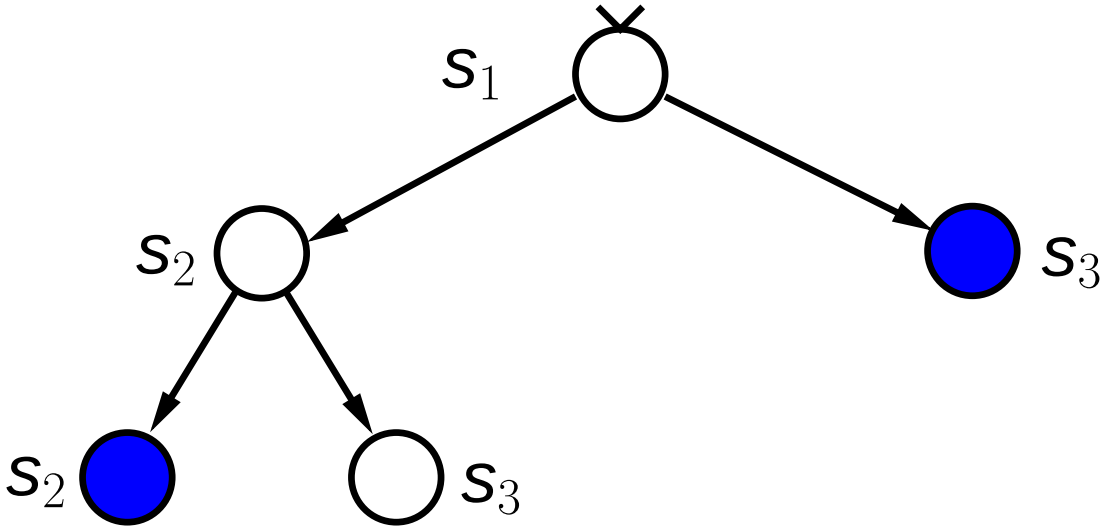
Example



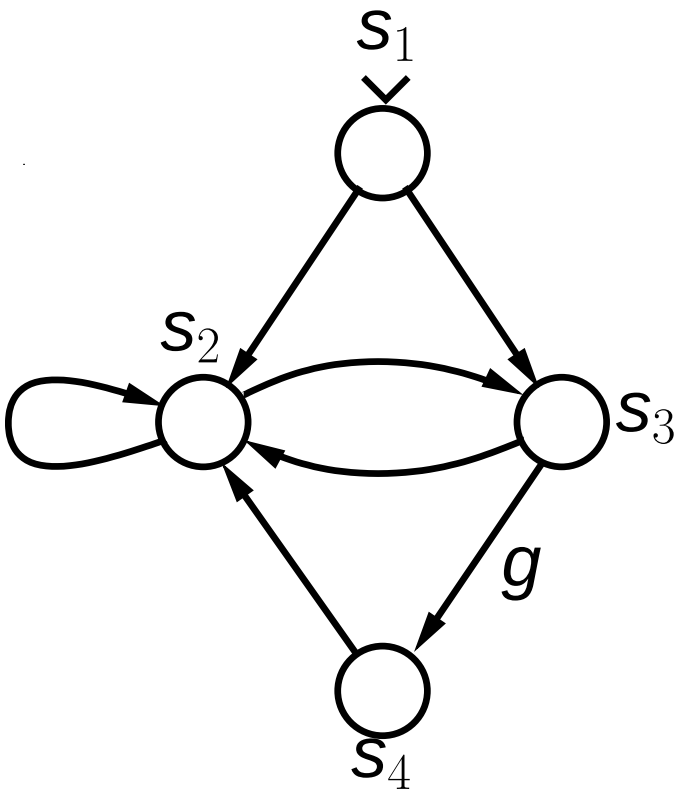


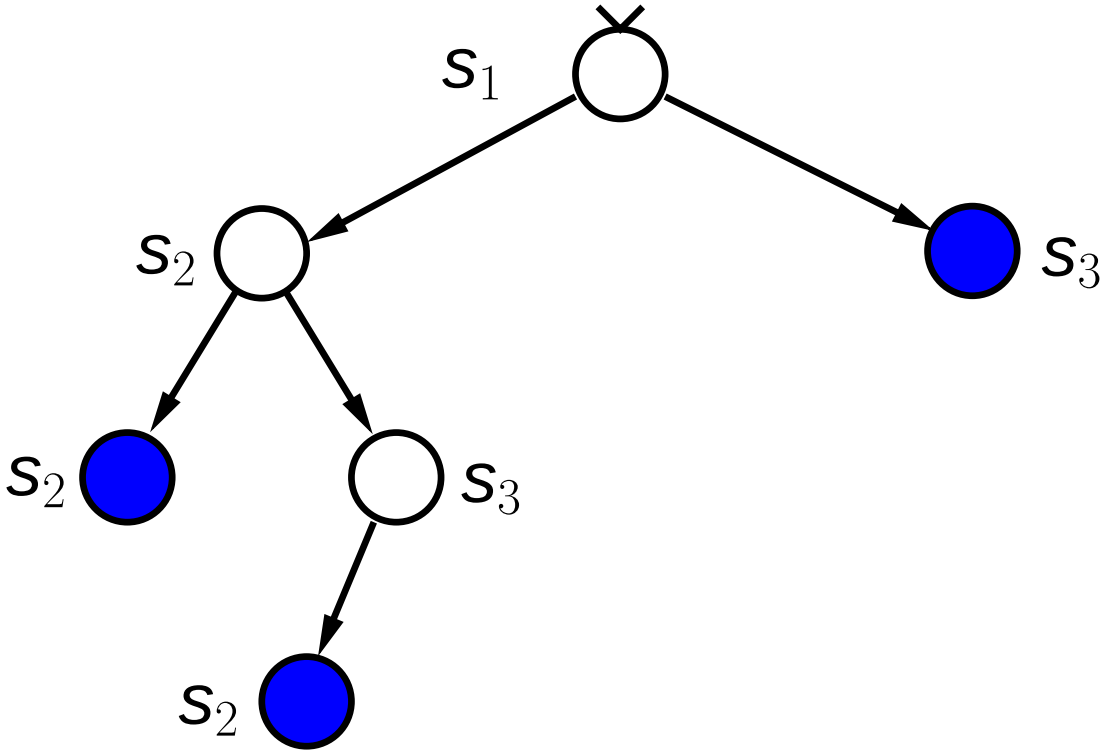
Example



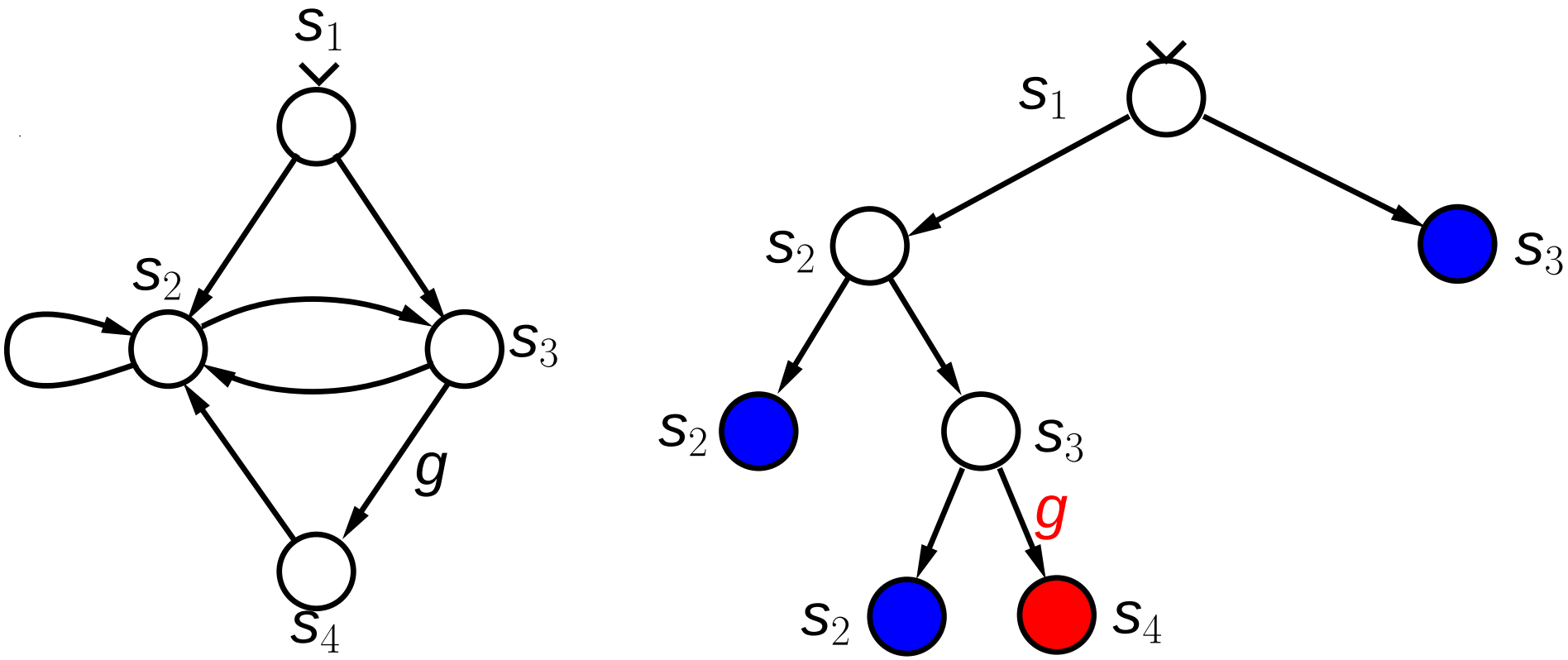


Example





Example



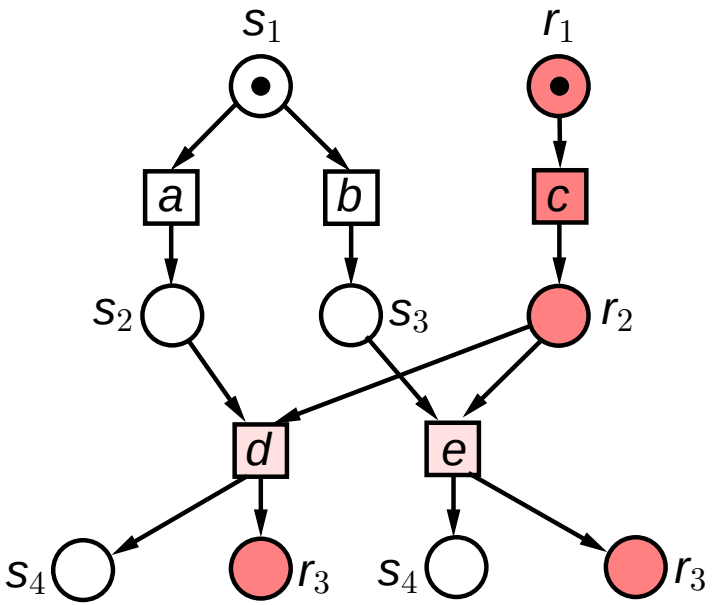
Searching the unfolding

We want something like this:

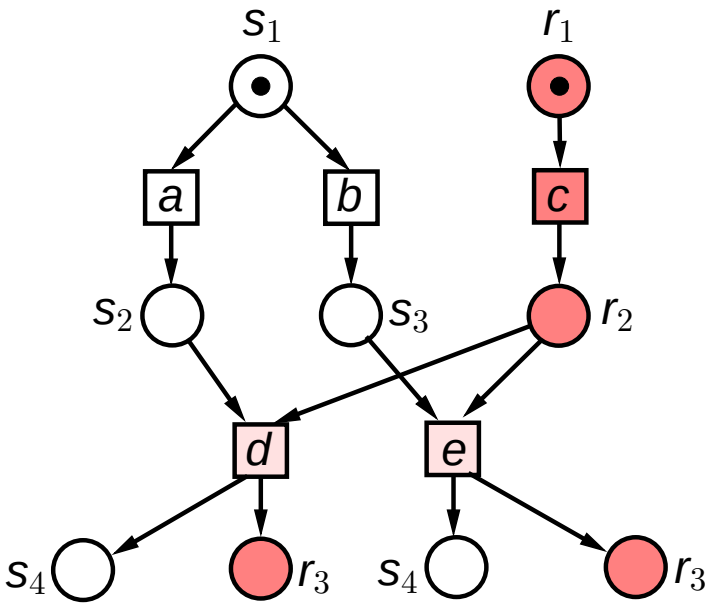
Termination condition: An event is a terminal if

- it is labeled by g or,
- "it leads to the same global state as another already executed event."

But an event of an unfolding does not usually lead to a unique global state



But an event of an unfolding does not necessarily lead to one global state

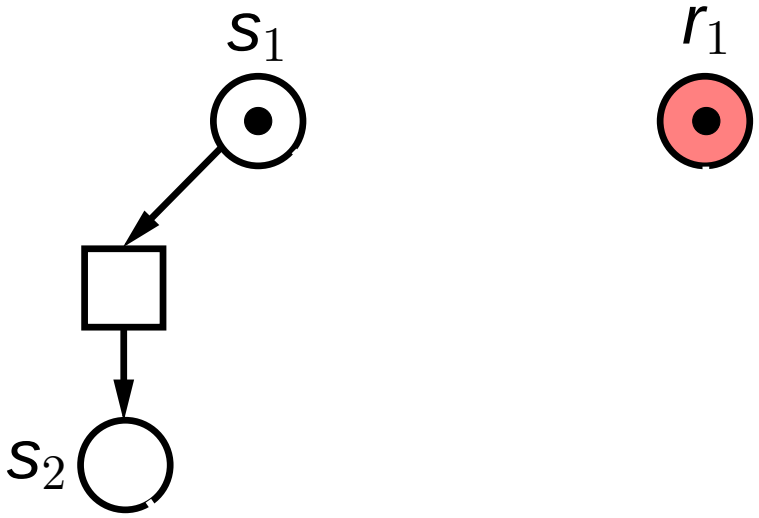


Solution (McMillan '92): attach to an event the global state reached by "executing its past".

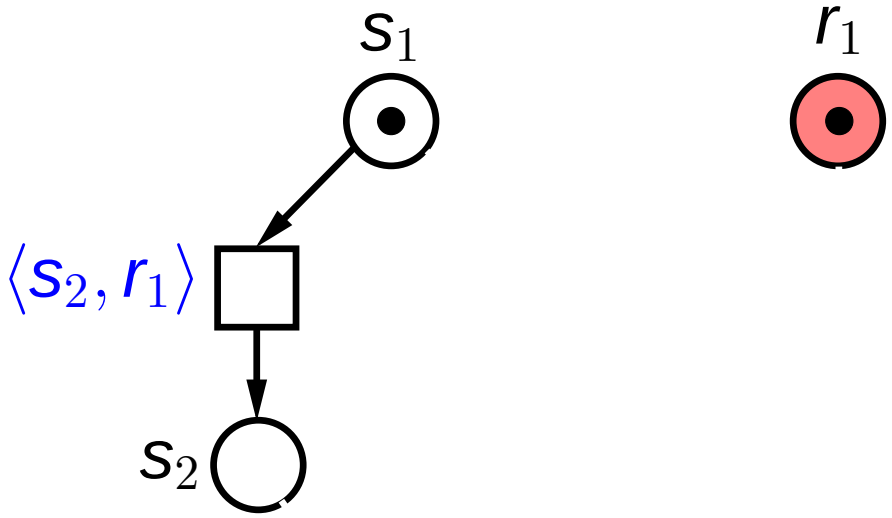
Call it the McMillan state

Example

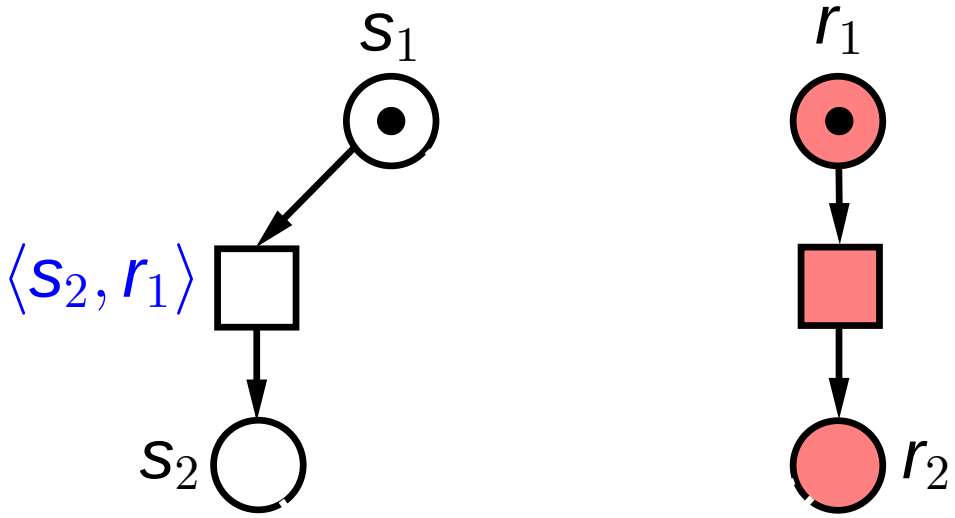
Example



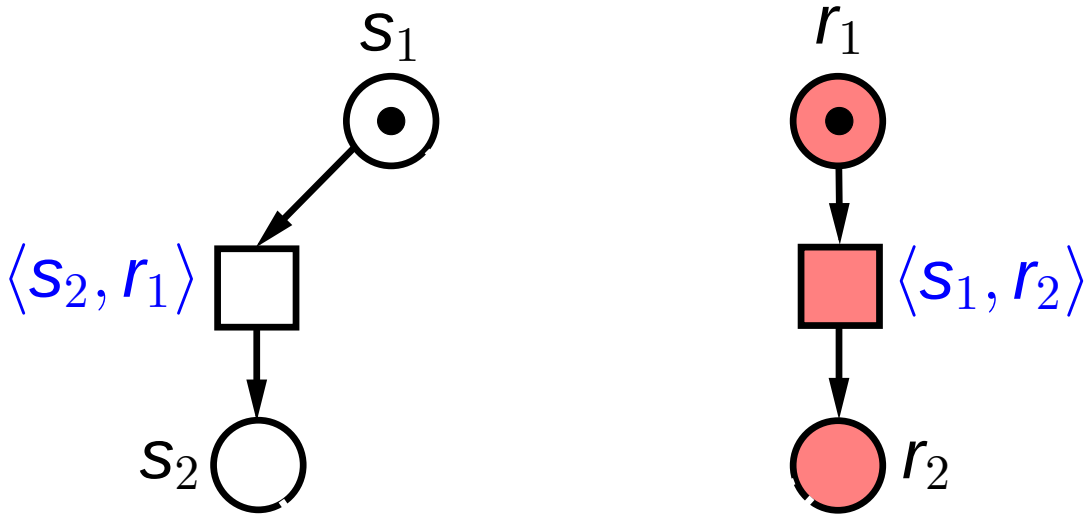
Example



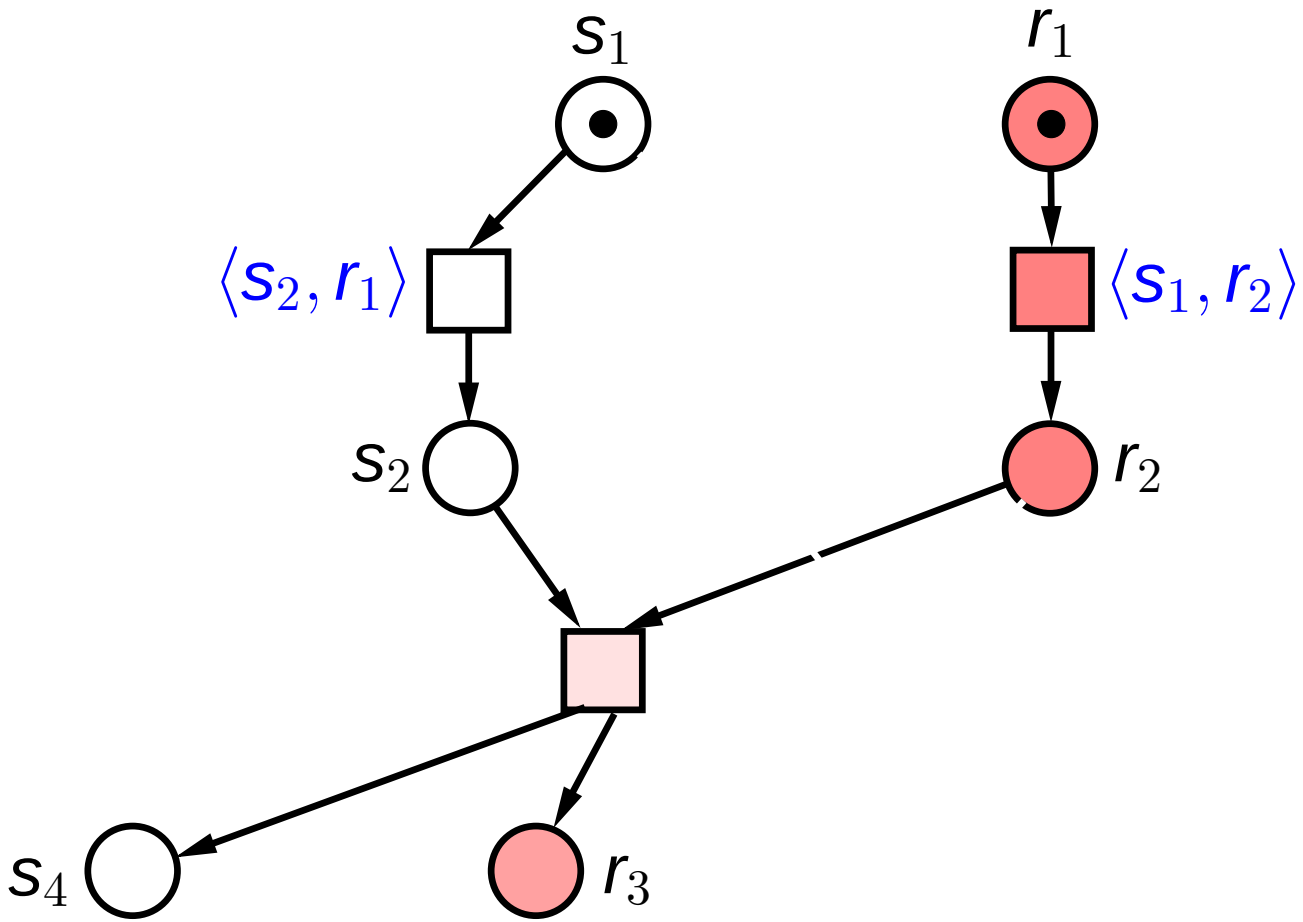
Example



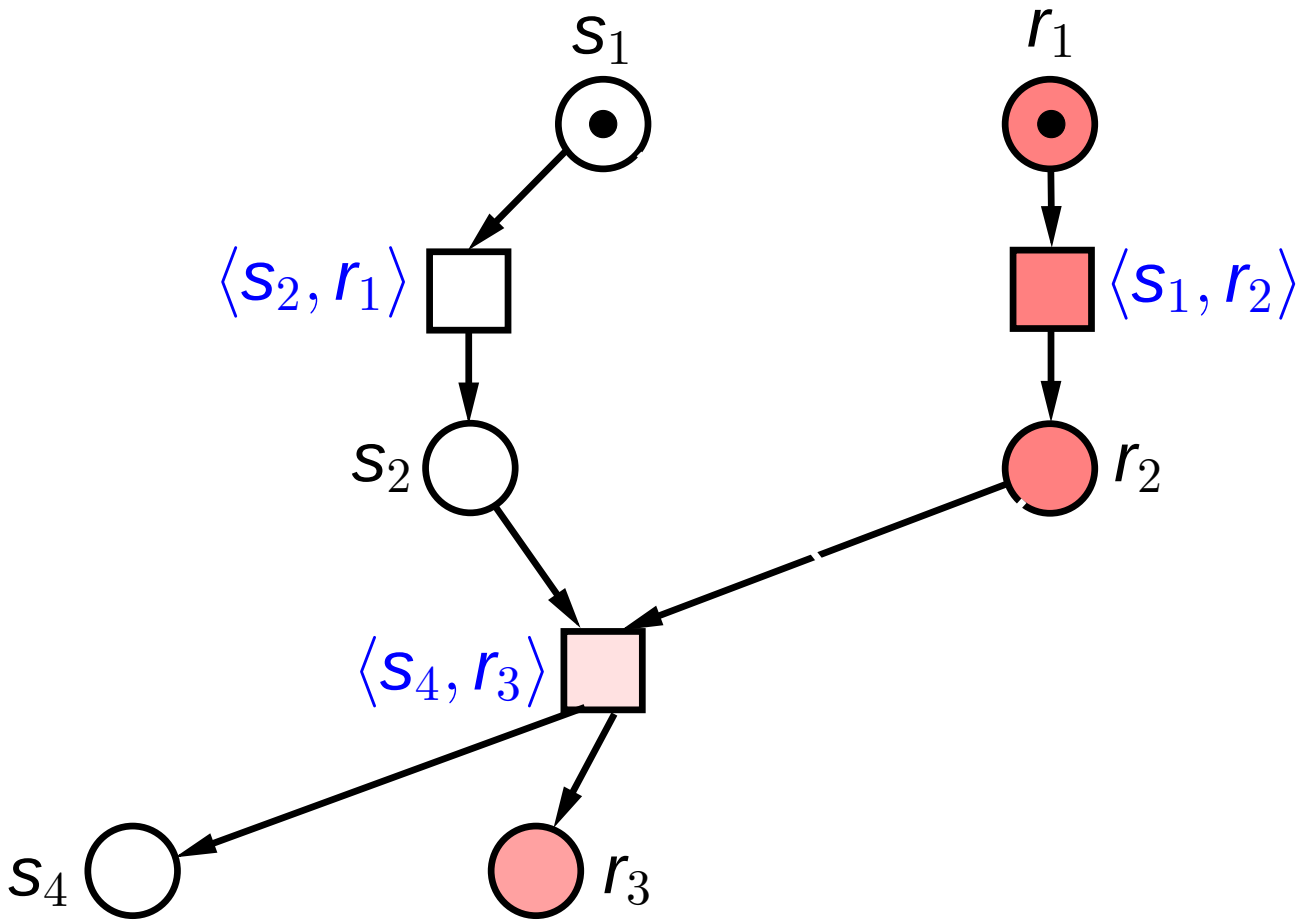
Example



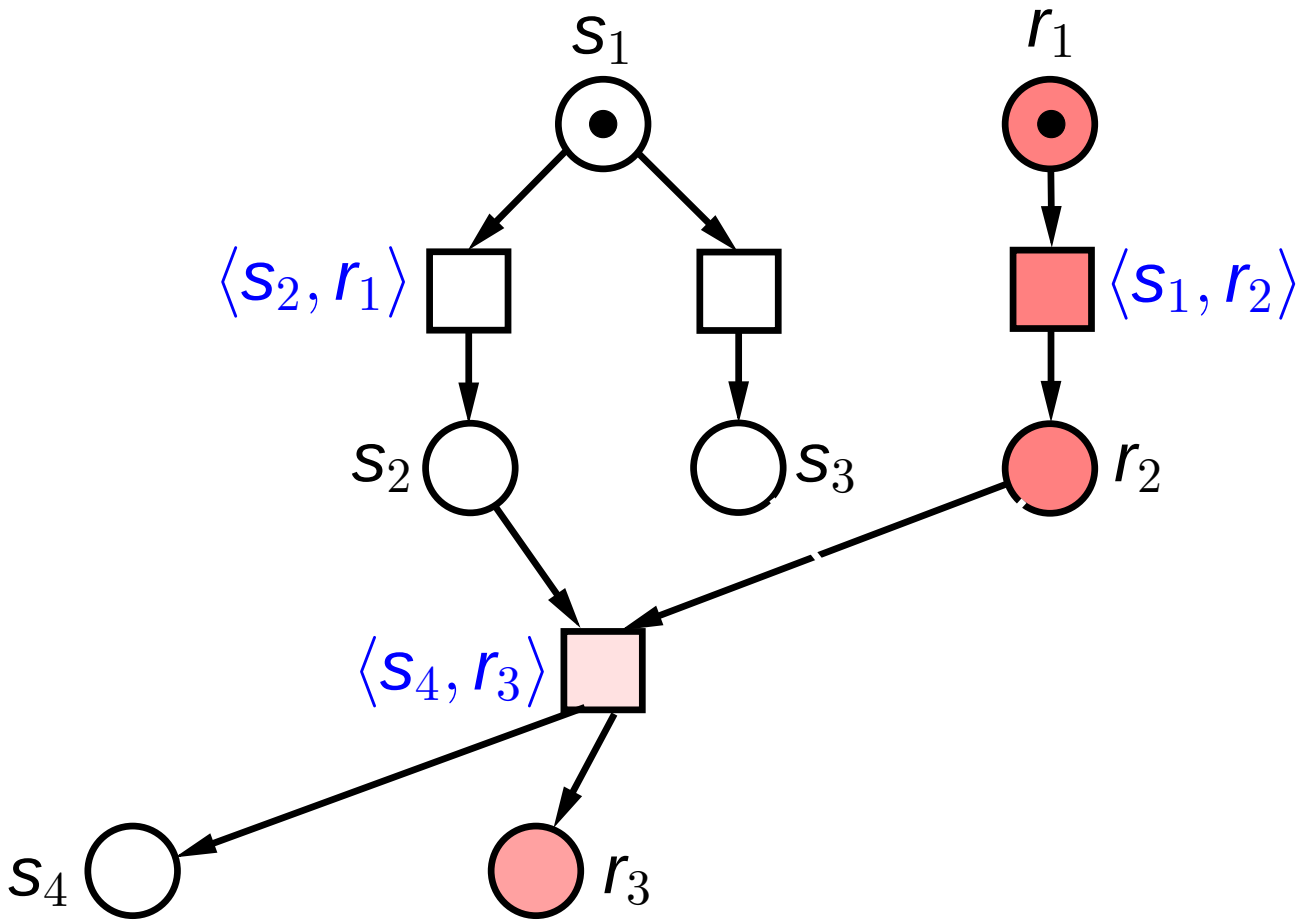
Example



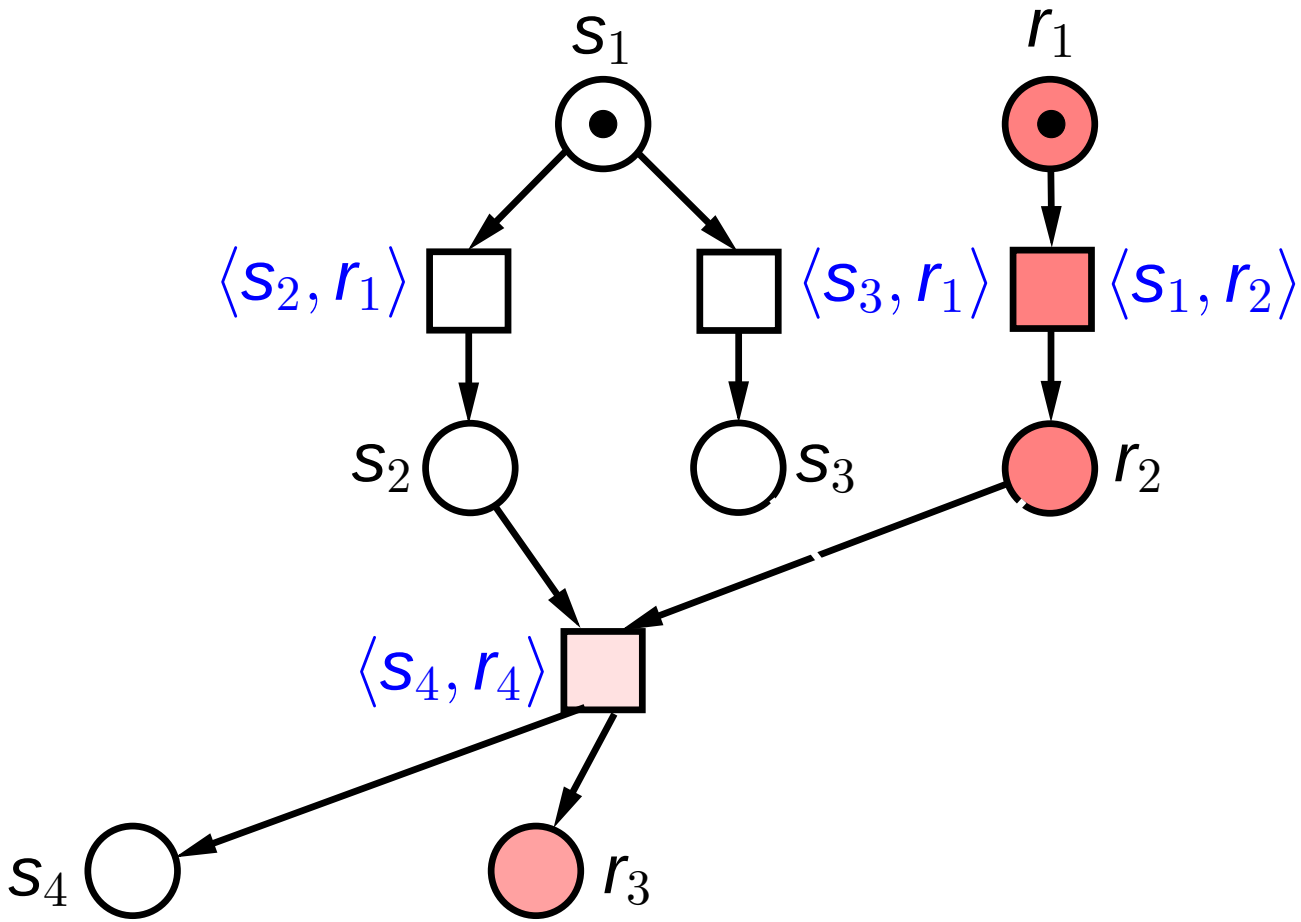
Example


s_1
r_1
$\langle s_2, r_1 \rangle$
$\langle s_1, r_2 \rangle$
s_2
r_2
$\langle s_4, r_3 \rangle$
s_4
r_3

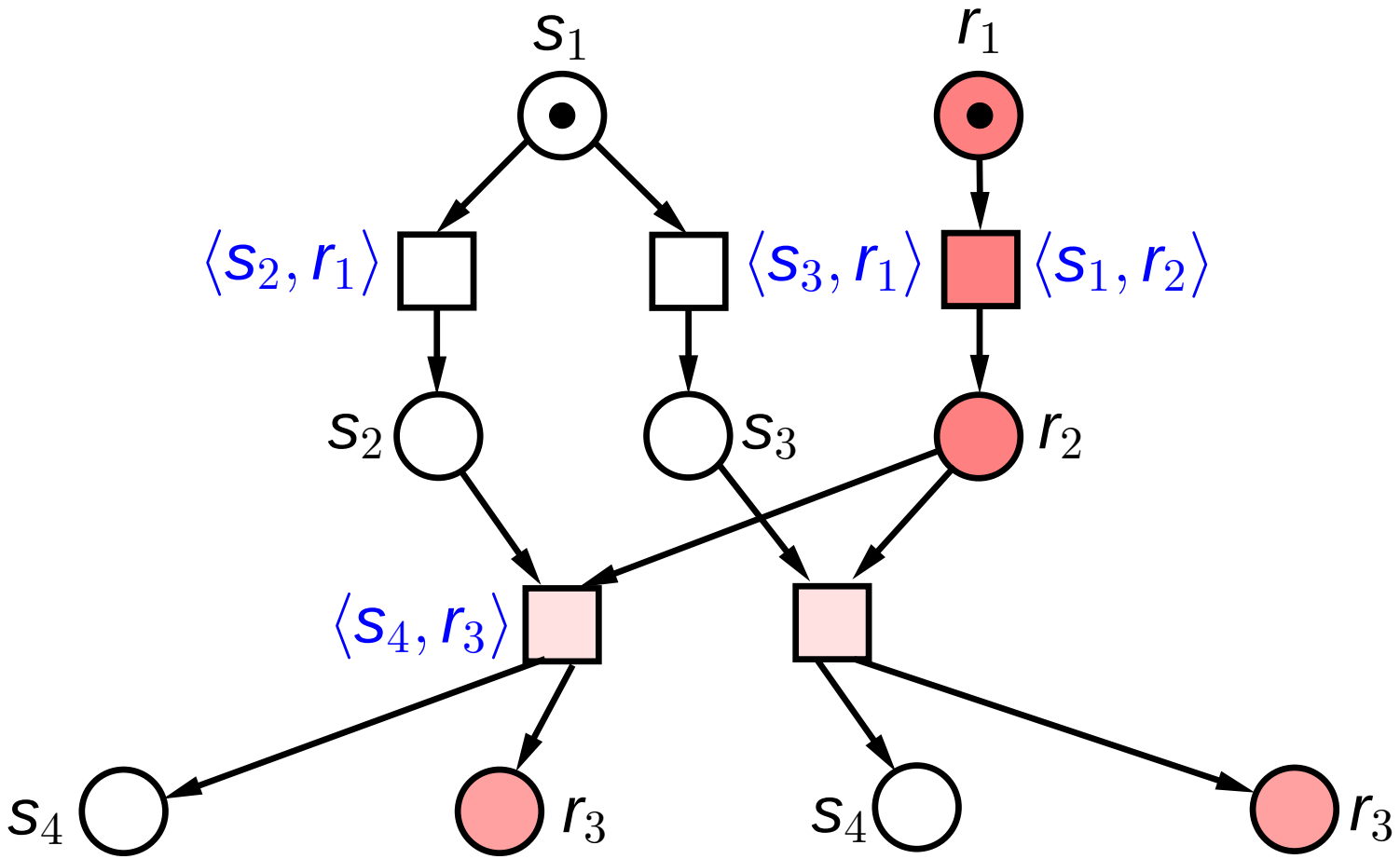
Example


s_1
r_1
$\langle s_2, r_1 \rangle$
$\langle s_1, r_2 \rangle$
s_2
s_3
r_2
$\langle s_4, r_3 \rangle$
s_4
r_3

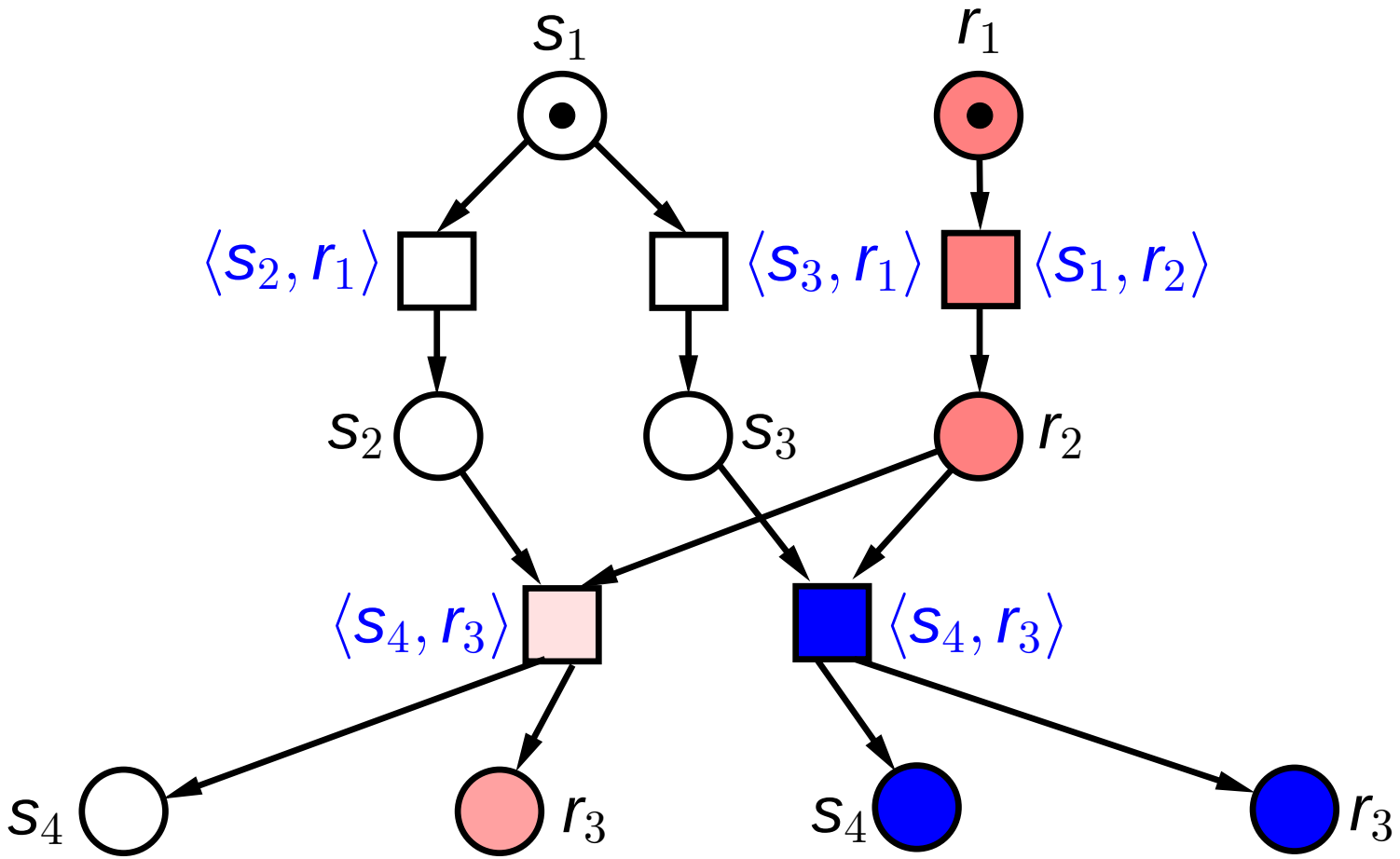
Example


s_1
r_1
$\langle s_2, r_1 \rangle$
$\langle s_3, r_1 \rangle$
$\langle s_1, r_2 \rangle$
s_2
s_3
r_2
$\langle s_4, r_4 \rangle$
s_4
r_3

Example


s_1
r_1
$\langle s_2, r_1 \rangle$
$\langle s_3, r_1 \rangle$
$\langle s_1, r_2 \rangle$
s_2
s_3
r_2
$\langle s_4, r_3 \rangle$
s_4
r_3
s_4
r_3

Example


s_1
r_1
$\langle s_2, r_1 \rangle$
$\langle s_3, r_1 \rangle$
$\langle s_1, r_2 \rangle$
s_2
s_3
r_2
$\langle s_4, r_3 \rangle$
$\langle s_4, r_3 \rangle$
s_4
r_3
s_4
r_3

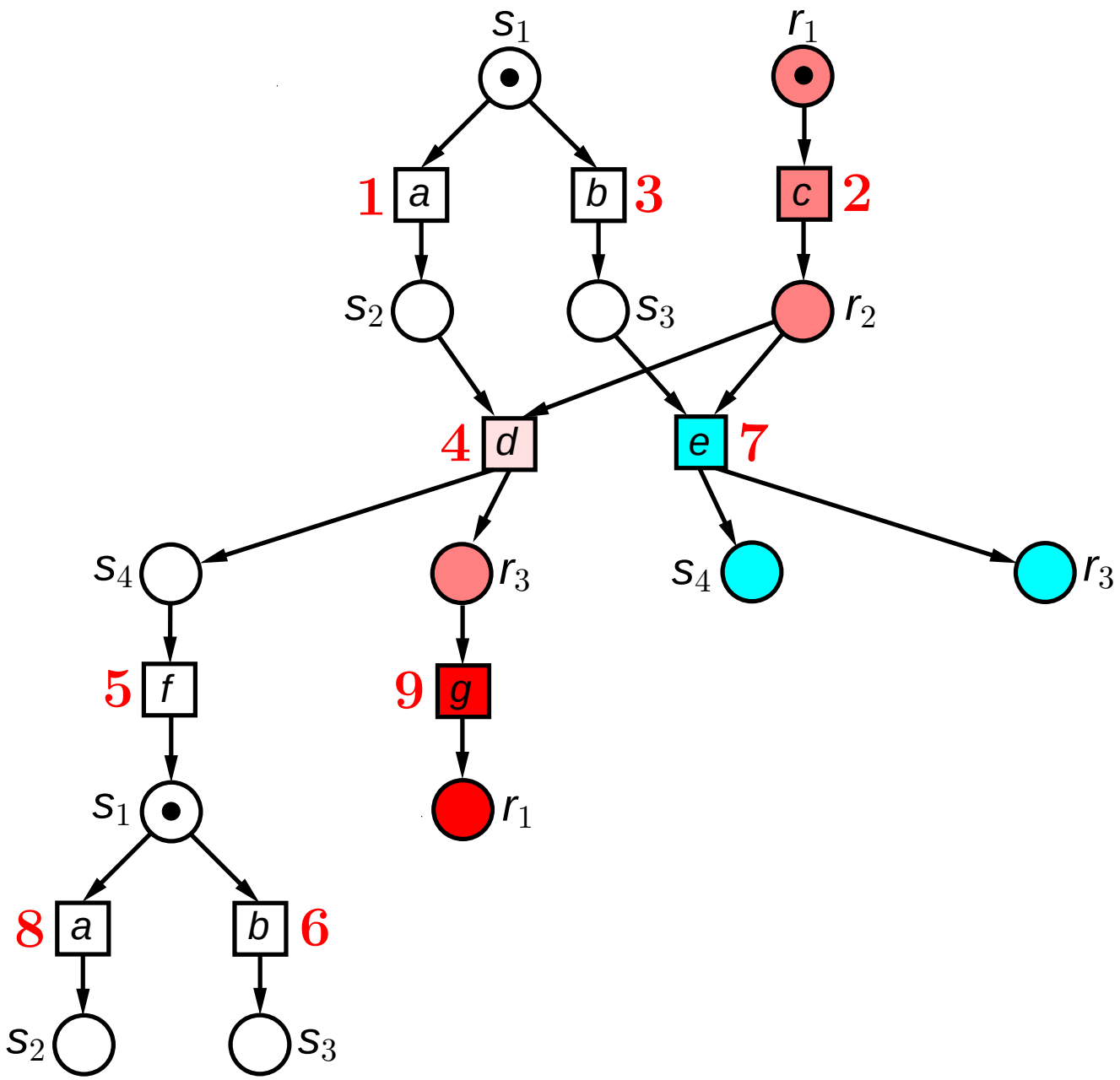
First attempt at a search algorithm

Search procedure to decide executability of g

Search strategy: any.

Termination condition: An event is a terminal if

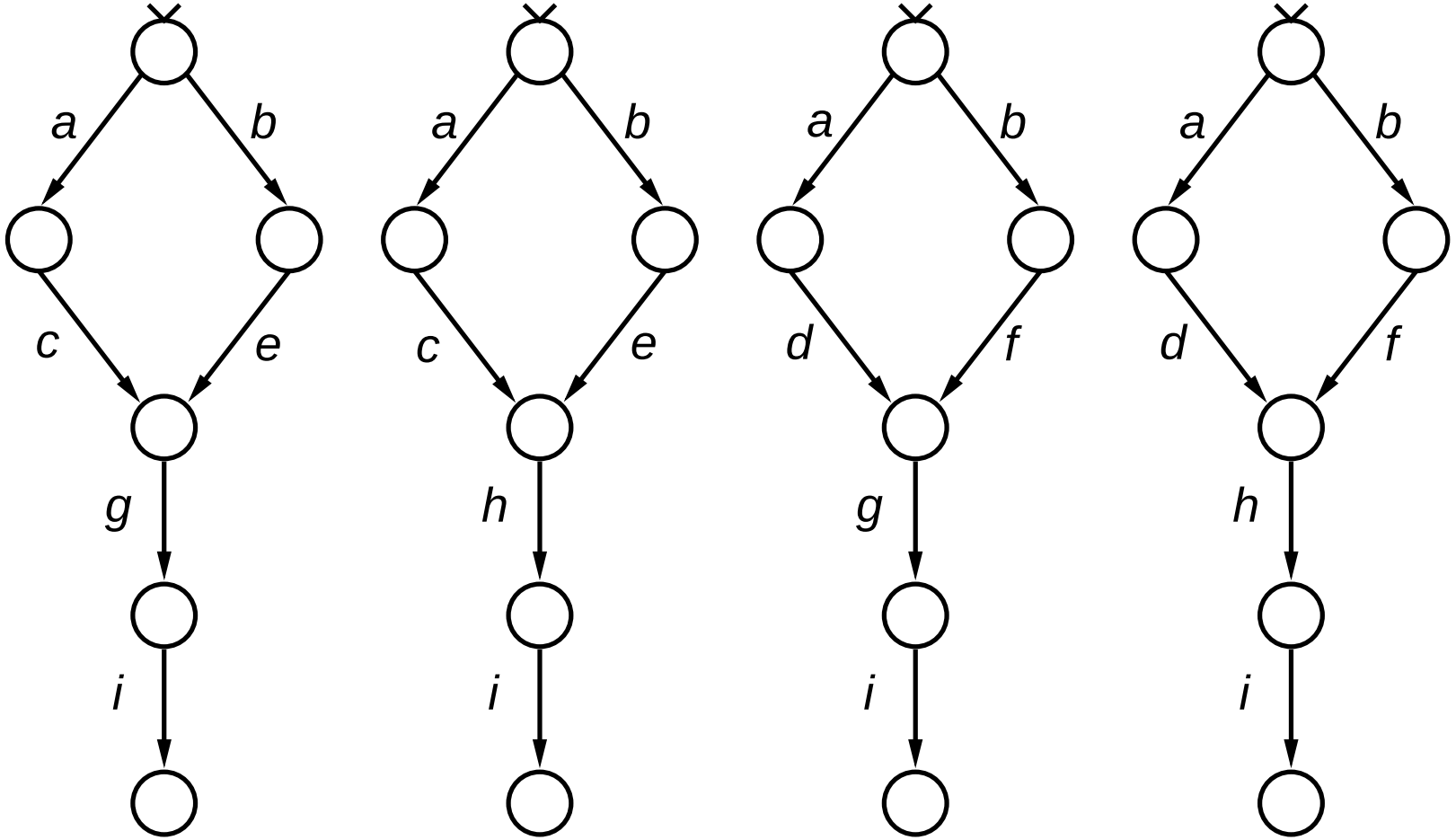
- it is labeled by g or,
- its McMillan state is the same as the McMillan state of another already explored event.

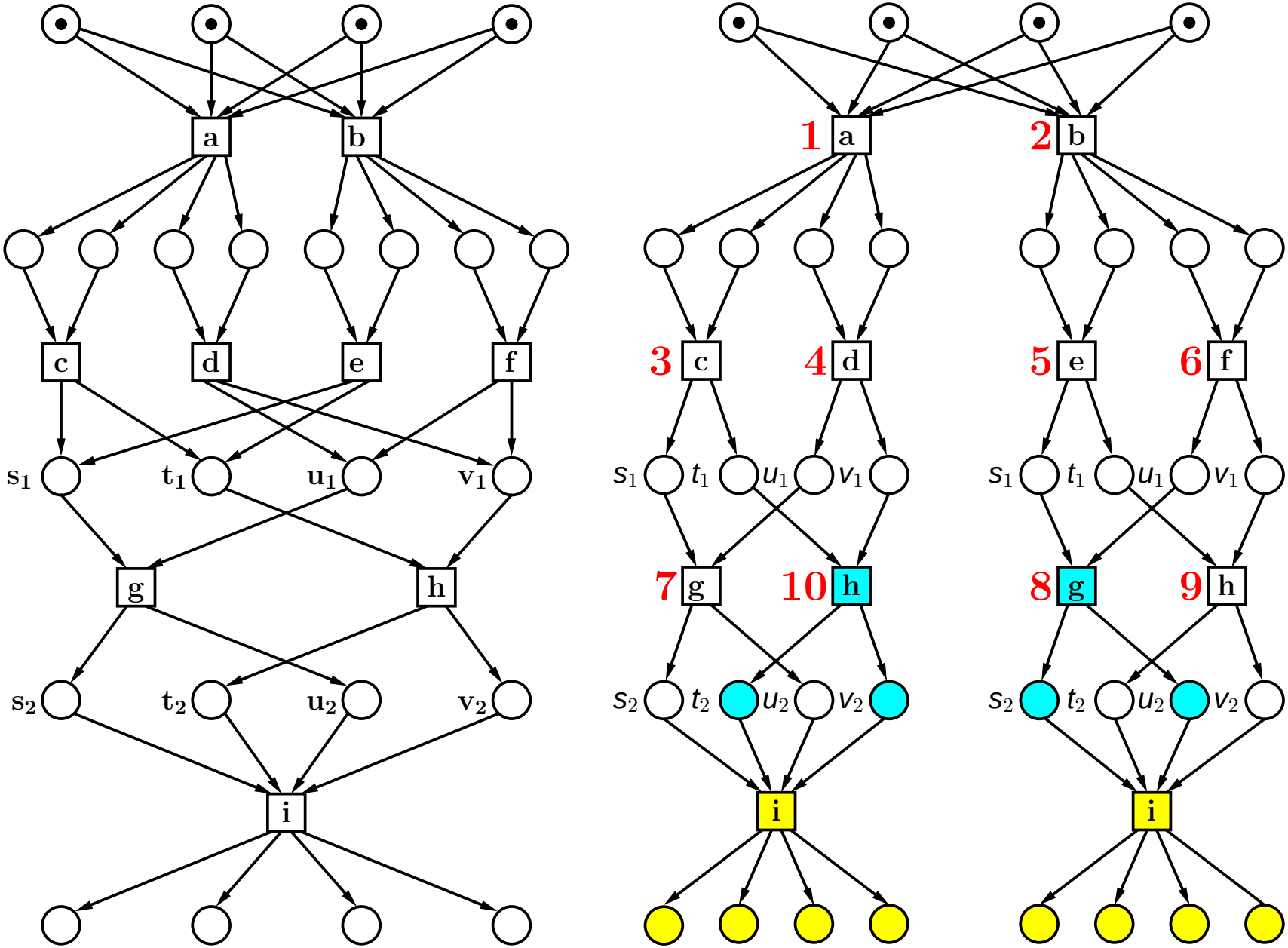

s_1
r_1
1 a
b 3
c 2
s_2
s_3
r_2
4 d
e 7
s_4
r_3
s_4
r_3
5 f
9 g
s_1
r_1
8 a
b 6
s_2
s_3

Is the search algorithm correct ?

Is the search algorithm correct ?

No !!


a
b
c
e
g
i
a
b
c
e
h
i
a
b
d
f
g
i
a
b
d
f
h
i


a
b
c
d
e
f
s1
t1
u1
v1
g
h
s2
t2
u2
v2
i
1
2
3
4
5
6
7
8
9
10

McMillan's solution

Stronger termination condition: An event e is a terminal if

- it is labeled by g or,
- its McMillan state is the same as the McMillan state of another, already executed, event e'

McMillan's solution

Stronger termination condition: An event e is a terminal if

- it is labeled by g or,
- its McMillan state is the same as the McMillan state of another, already executed, event e'
  and the past of e' contains strictly fewer events than the past of e.

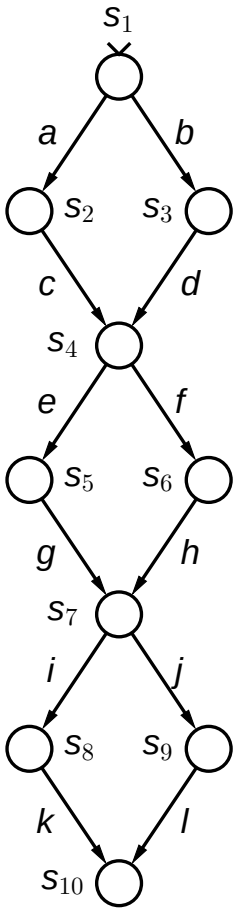
McMillan's solution

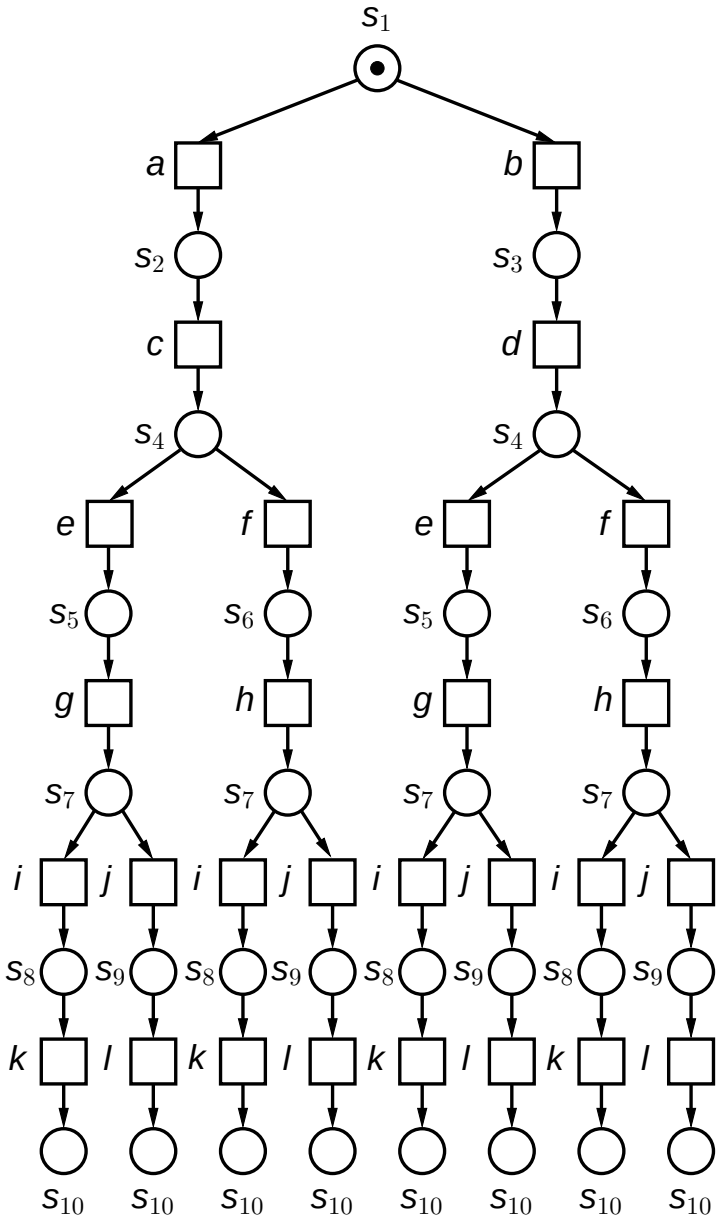
Stronger termination condition: An event e is a terminal if

- it is labeled by g or,
- its McMillan state is the same as the McMillan state of another, already executed, event e'
  and the past of e' contains strictly fewer events than the past of e.

Downside: stronger condition $\longrightarrow$ fewer terminals $\longrightarrow$ larger prefix

Worst case: prefix exponentially larger than number of reachable global states.


s_1
a
b
s_2
s_3
c
d
s_4
e
f
s_5
s_6
g
h
s_7
i
j
s_8
s_9
k
l
s_{10}


s_1
a
b
s_2
s_3
c
d
s_4
s_4
e
f
e
f
s_5
s_6
s_5
s_6
g
h
g
h
s_7
s_7
s_7
s_7
i
j
i
j
i
j
i
j
s_8
s_9
s_8
s_9
s_8
s_9
s_8
s_9
k
l
k
l
k
l
k
l
s_{10}
s_{10}
s_{10}
s_{10}
s_{10}
s_{10}
s_{10}
s_{10}

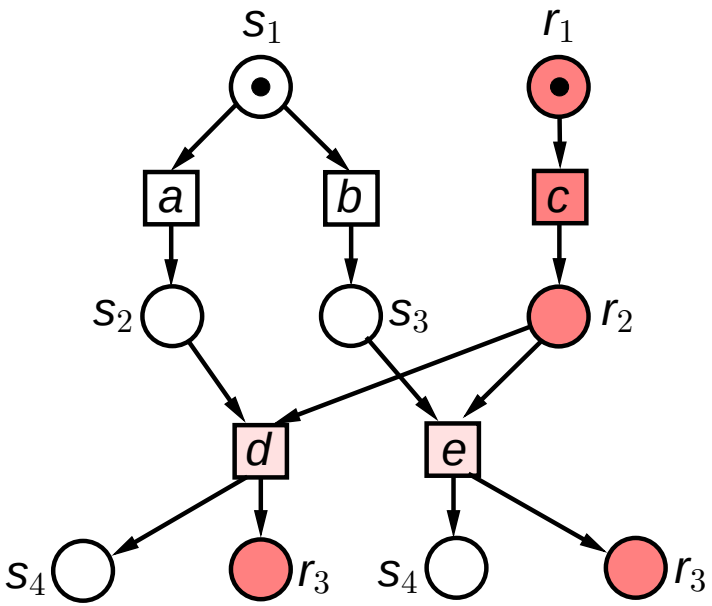
1996

Adequate strategies

A better solution by E., Römer, and Vogler.

General idea: Keep the termination condition, restrict the strategies.

The past of an event can be seen as a Mazurkiewicz trace: the set of all possible ways of executing the past.



Adequate strategies

Selecting the next event amounts to selecting its Mazurkiewicz trace.

Adequate strategies

Selecting the next event amounts to selecting its Mazurkiewicz trace.

Strategy $\longrightarrow$ total order $\prec$ on Mazurkiewicz traces.

Adequate strategies

Selecting the next event amounts to selecting its Mazurkiewicz trace.

Strategy $\longrightarrow$ total order $\prec$ on Mazurkiewicz traces.

A strategy $\prec$ on Mazurkiewicz traces is adequate if

$$t \prec t' \quad \text{implies} \quad t\,t'' \prec t'\,t'' \quad \text{for all } t''.$$

Are there adequate strategies?

Fact 1: every adequate strategy on executions can be "lifted" to an adequate strategy on Mazurkiewicz traces.

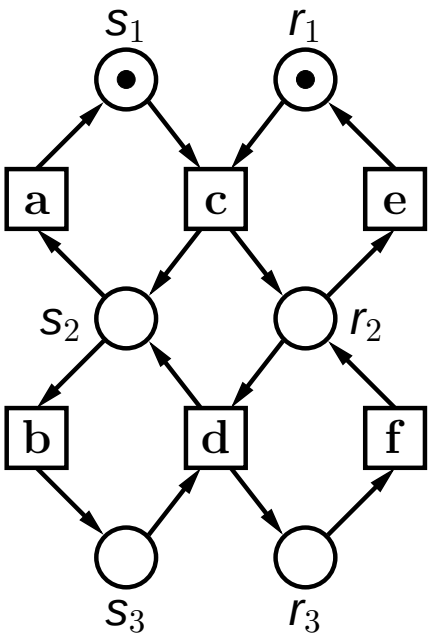
Fact 2: the following strategy on executions is adequate: $w_1 \prec w_2$ iff

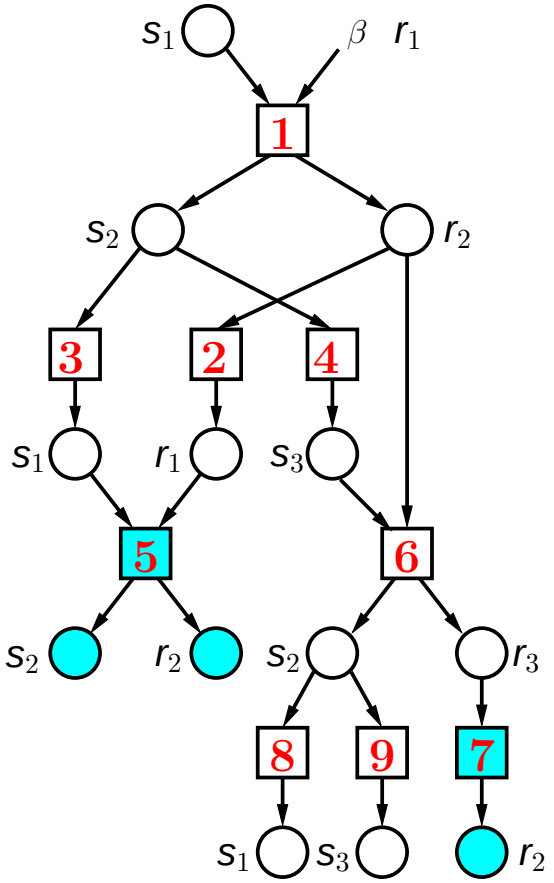
- $|w_1| < |w_2|$, or;
- $|w_1| = |w_2|$ and w_1 is lexicographically smaller than w_2.

Many adequate strategies have been found: E., Römer, Vogler '96, E., Römer '99, Niebert, Qu '06.

Complete prefixes

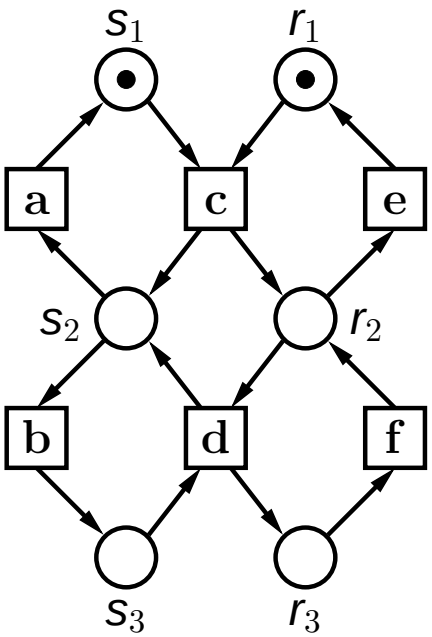
Prefixes of the unfolding containing all reachable states.

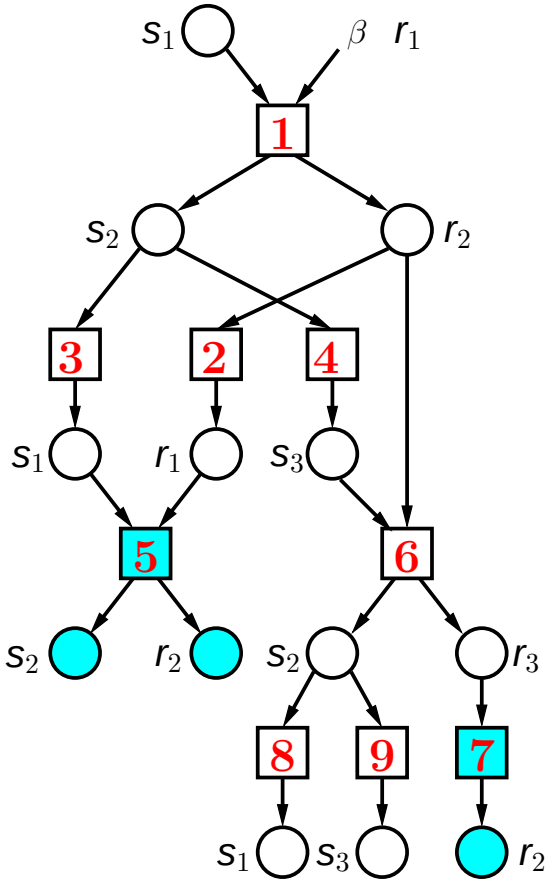




Complete prefixes

Prefixes of the unfolding containing all reachable states.





Compact representations of the state space.

Complete prefixes

System (scale)	Structured Petri net			Prefix		BDD size (Petrify)
	Places	Trans.	States	Places	Trans.	
Cycl(12)	95	71	74264	232	104	
PhilD(7)	63	63	109965	86310	4314	
SIRing(10)	100	100	8.1×10^{12}	119450	86180	
Elevator(4)	736	1939	43440	32354	16935	
ProdCell	231	202	$> 3.1 \times 10^{6}$	2164	1035	40188
ConcPush	150	140	2.8×10^{7}	1671	780	210249
DMut(64)	257	256	1.8×10^{62}	385	256	45460
RW(10)	86	66	1.6×10^{6}	29132	15974	7576

100 random tables with right-handed, left-handed, and ambidextrous philosophers

BDD for the set of reachable states (Petrify)

Nr. of phil.	BDD size				
	Average	Min.	Max.	St.Dev.	Aver./St.Dev.
4	178	94	355	52	0.30
6	583	248	1716	305	0.52
8	1553	390	8678	1437	0.92
10	3140	510	27516	4637	1.48
12	4855	632	47039	8538	1.76
14	33742	797	429903	85798	2.54

100 random tables with right-handed, left-handed, and ambidextrous philosophers

Nodes of the complete prefix (PEP)

Nr. of phil.	Prefix size				
	Average	Min.	Max.	St.Dev.	Aver./St.Dev.
4	46	40	60	5.13	0.10
6	70	60	85	5.99	0.09
8	95	80	110	6.92	0.07
10	117	100	135	7.78	0.07
12	141	120	160	7.40	0.05
14	161	140	185	9.25	0.06

Complete prefixes

Prefixes of the unfolding containing all reachable states.

Compact representations of the state space.

But the net itself is already an (even more) compact representation!

Trade-off between compactness and query complexity.

Complete prefixes

Prefixes of the unfolding containing all reachable states.

Compact representations of the state space.

But the net itself is already an (even more) compact representation!

Trade-off between compactness and query complexity.

Reachability of a global state:

Petri net	Complete prefix	Transition system
PSPACE-complete		Linear

Complete prefixes

Prefixes of the unfolding containing all reachable states.

Compact representations of the state space.

But the net itself is already an (even more) compact representation!

Trade-off between compactness and query complexity.

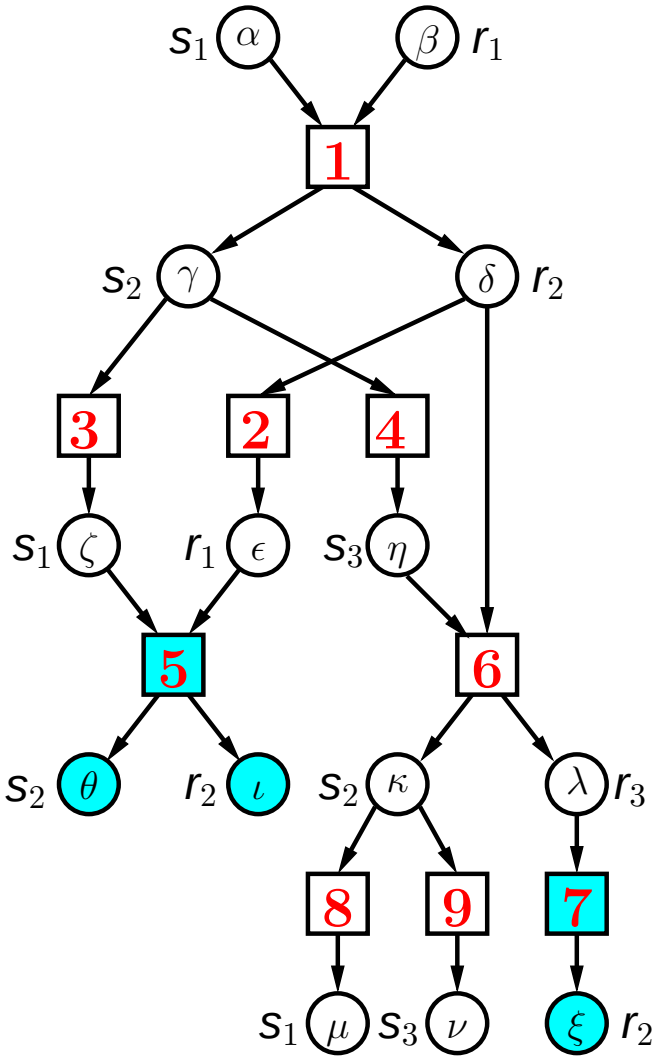
Reachability of a global state:

Petri net	Complete prefix	Transition system
PSPACE-complete	NP-complete	Linear

1999

Querying a prefix with SAT

First studied by Keijo Heljanko in his thesis and several papers.



place	clause
$\boldsymbol{\alpha}$	$\boldsymbol{\alpha} \leftrightarrow \neg \mathbf{1}$
$\boldsymbol{\beta}$	$\boldsymbol{\beta} \leftrightarrow \neg \mathbf{1}$
$\boldsymbol{\gamma}$	$(\,(\mathbf{3} \vee \mathbf{4}) \rightarrow \mathbf{1}) \wedge \neg(\mathbf{3} \wedge \mathbf{4})$
	$\wedge(\boldsymbol{\gamma} \leftrightarrow (\mathbf{1} \wedge \neg \mathbf{3} \wedge \neg \mathbf{4})\,)$
$\boldsymbol{\delta}$	$(\,(\mathbf{2} \vee \mathbf{6}) \rightarrow \mathbf{1}) \wedge \neg(\mathbf{2} \wedge \mathbf{6})$
	$\wedge(\boldsymbol{\delta} \leftrightarrow (\mathbf{1} \wedge \neg \mathbf{2} \wedge \neg \mathbf{6})\,)$
$\ldots$	
$\boldsymbol{\xi}$	$\boldsymbol{\xi} \leftrightarrow \mathbf{9}$

Checking deadlock-freedom with BDDs

100 random tables with right-, left-handed, and ambidextrous philosophers

SMV on a SUN Ultra 60, 2 processors, 640 MB (old experiment)

Nr of phil.	Time in seconds				
	Average	Min.	Max.	St.Dev.	Aver./St.Dev.
4	0.08	0.05	0.13	0.02	0.29
6	0.36	0.20	1.18	0.16	0.46
8	4.14	1.25	14.60	2.45	0.59
10	56.60	15.80	388.00	46.90	0.83
12	1595.00	228.00	10616.00	1615.00	1.01

Checking deadlock-freedom with unfoldings

100 random tables with right-, left-handed, and ambidextrous philosophers

PEP + stable models on a SUN Ultra 60, 2 processors, 640 MB (old experiment)

Nr. of phil.	Time in seconds				
	Average	Min	Max	St. Dev	Aver./St. Dev
8	0.01	0.04	0.03	0.007	0.24
10	0.01	0.06	0.03	0.009	0.27
12	0.02	0.07	0.04	0.012	0.28
14	0.02	0.05	0.04	0.007	0.20
16	0.02	0.05	0.04	0.007	0.17
18	0.03	0.05	0.04	0.007	0.17

2000-2010

2000-2010


Harry Potter
À
L'ÉCOLE DES
SORCIERS


Harry Potter
AND THE ORDER OF THE PHOENIX

2000-2010

A quick summary

• Efficient construction of complete prefixes, adequate strategies.
Chatain, E., Khomenko, Koutny, Niebert, Schwoon, Vogler ...

• Full LTL model-checking.
Couvreur, E., Heljanko, Grivet, Poitrenaud ...

• Parallel and distributed generation of the unfolding.
Baldan, Haar, Heljanko, Khomenko, König, Koutny ...

• Extensions to other concurrency models.
Baldan, Corradini, Haar, Khomenko, König, Langerak, Meyer, Schröter, Schwoon ...

• Extensions to timed models.
Bouyer, Cassez, Chatain, Haddad, Jard ...

Applications I: Asynchronous circuits

Circuits specified as Signal Transition Graphs (interpreted Petri nets)

Transitions raise and lower boolean signals

Correctness: every global state completely determined by the values of the signals

Concurrent Asynchronous Systems Group, University of Newcastle: tool-chain for verification and fault-fixing of STGs based on unfoldings.

Khomenko, Schäfer, Vogler, Yakovlev, Wollowski ...

Applications II: Monitoring and diagnosis

Distributed systems with sensors attached to some nodes

Alarms reported to global supervisor, who performs diagnosis

Problem: find cause of the alarms $\longrightarrow$ true-concurrency approach

IRISA group in Rennes, MEXICO project at ENS Cachan: diagnosis tools.

Benveniste, Chatain, Haar, Jard ...

Applications III: Graph Grammars

Unfolding used to overapproximate the set of graphs generated by graph transformation systems

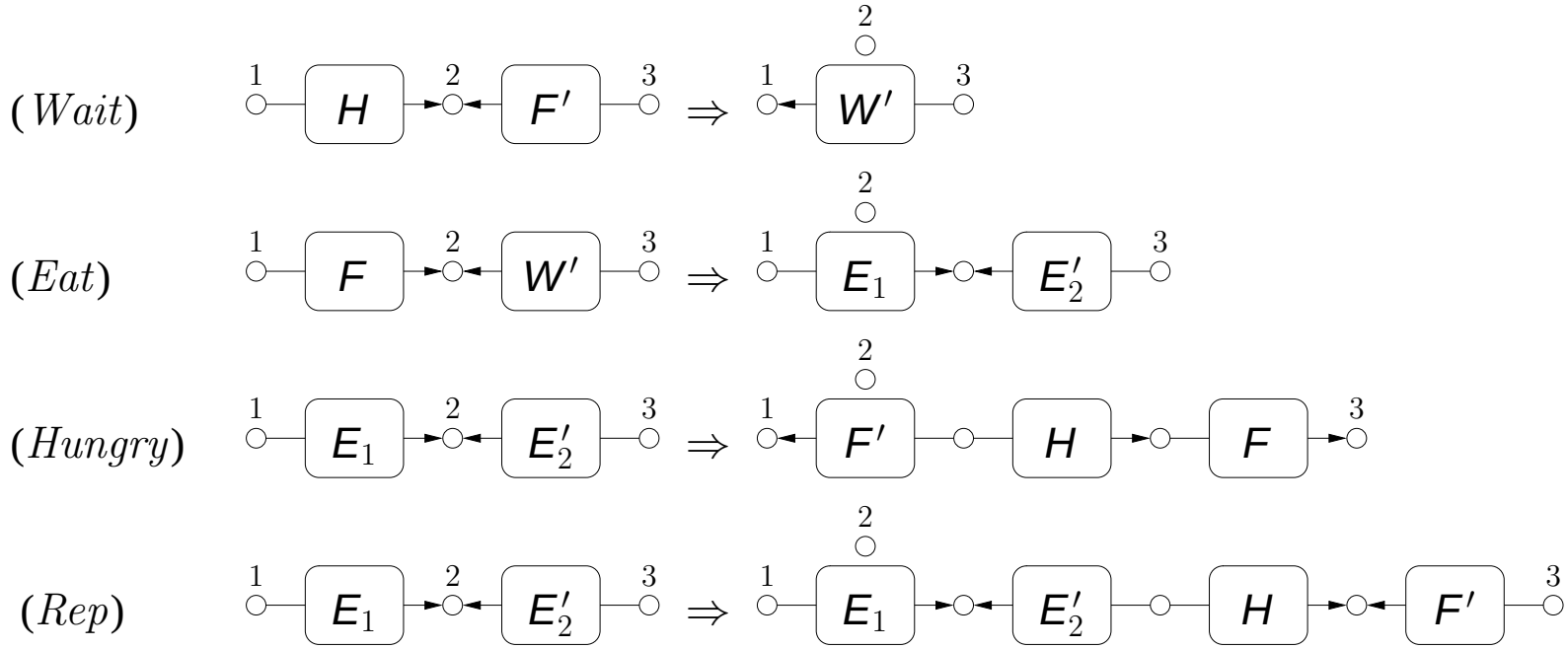
"Folding the unfolding" yields an unbounded Petri net, with each marking representing a graph.

Model-checking algorithms based on the overapproximation

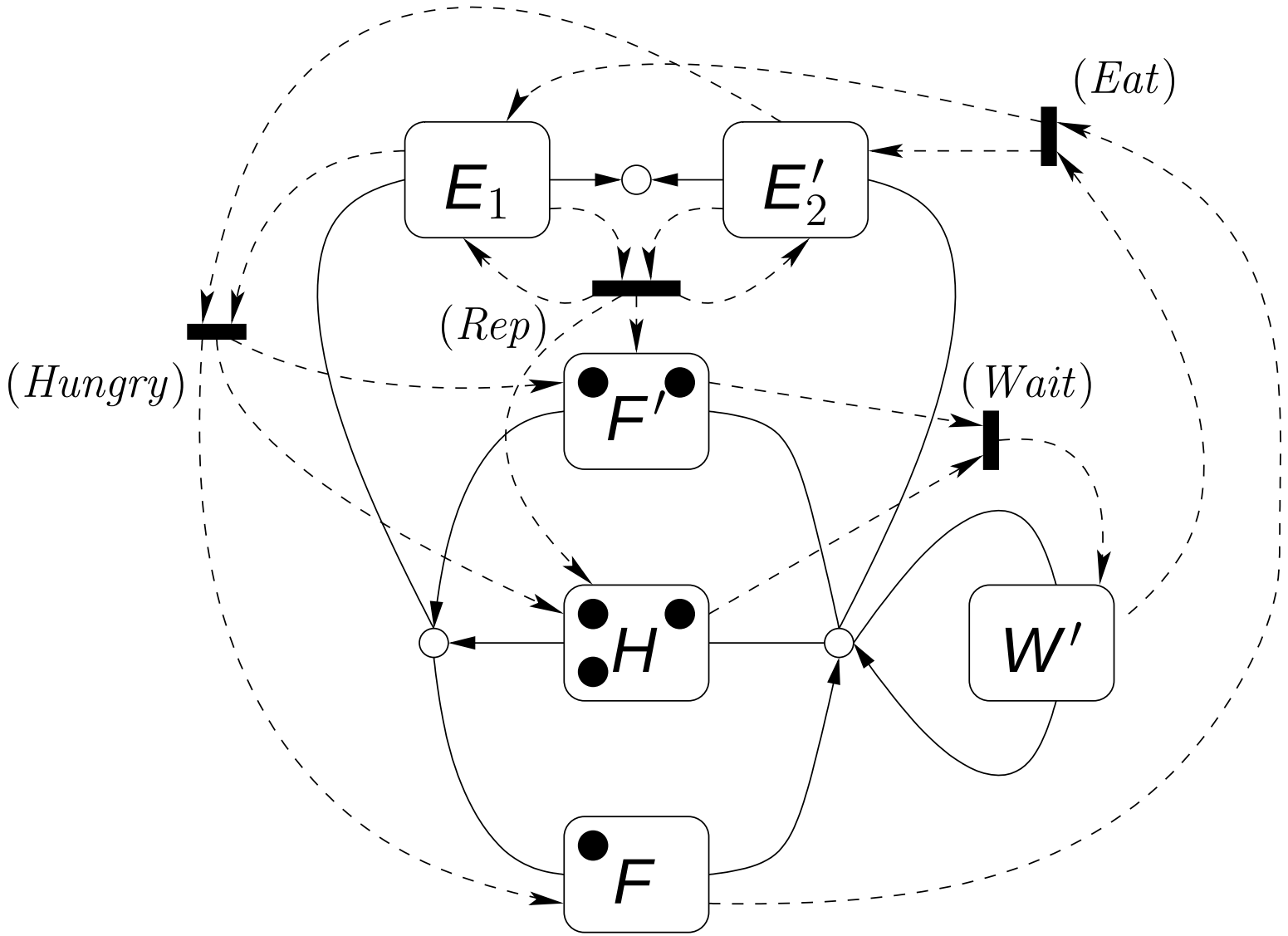
AUGUR 1 and 2.

Baldan, Corradini, König, Kozioura ...

Applications III: Graph Grammars



Applications III: Graph Grammars



Conclusions

From abstract connections between Physics and Computer Science to concrete algorithms and applications.

Turning point: verification through explicit construction of semantic objects.

True-concurrency useful in two ways:

- Compact representation of state spaces.
- Information about causality (diagnosis) and independence (probabilistic systems)