An Unfolding Approach to Model Checking

Javier Esparza

Laboratory for Foundations of Computer Science
University of Edinburgh

Concurrent programs

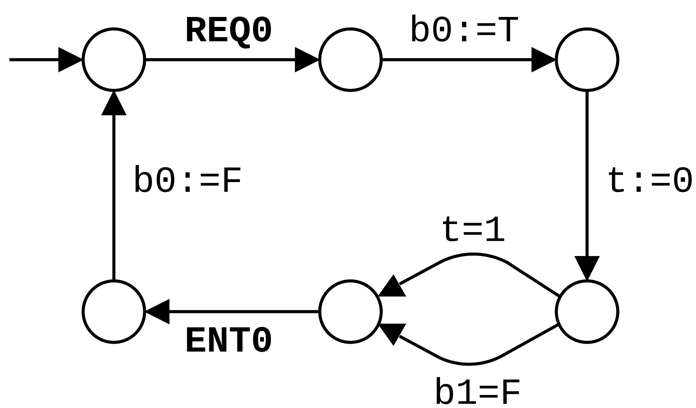
Program: a tuple $P = (T_1, \ldots T_n)$ of finite labelled transition systems

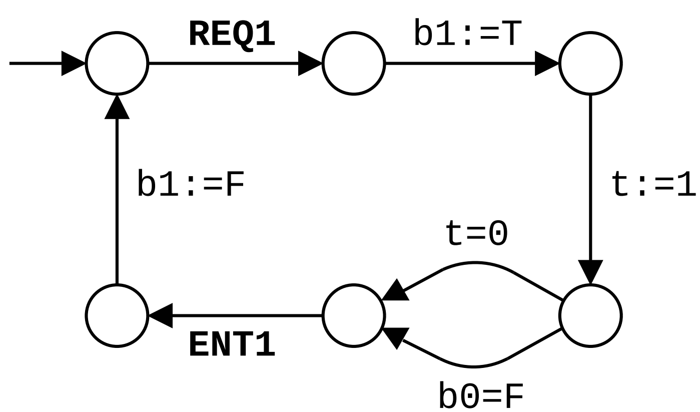
$$T_i = (A_i, S_i, \Delta_i, s_{0i}) \ , \quad 1 \leq i \leq n$$

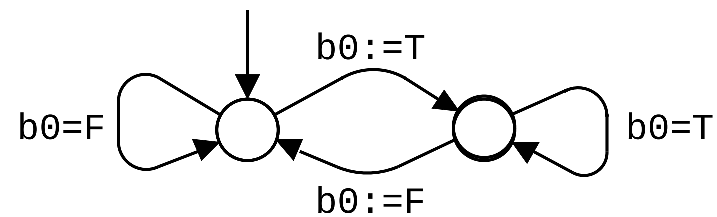
where

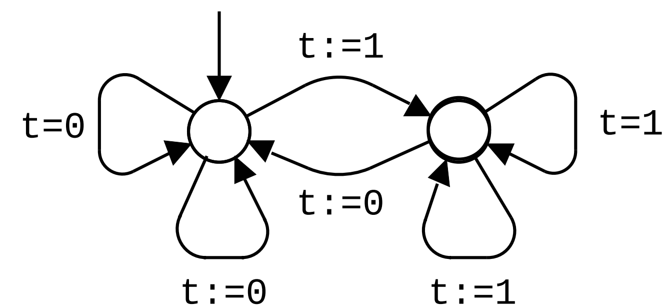
- A_i is an alphabet of actions,
- S_i is a finite set of (local) states,
- $\Delta_i \subseteq S_i \times A_i \times \S_i$ is a transition relation, and
- $s_{0i} \in S_i$ is the initial state.

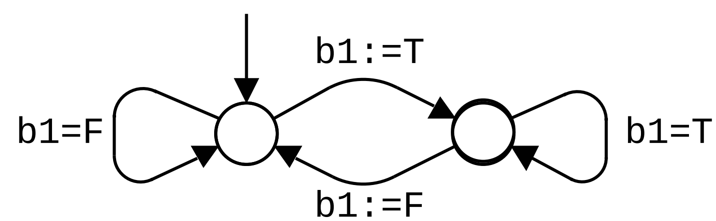
Example


REQ0
b0:=T
t:=0
t=1
b1=F
ENT0
b0:=F


REQ1
b1:=T
t:=1
t=0
b0=F
ENT1
b1:=F


b0:=T
b0=F
b0=T
b0:=F


t:=1
t=0
t=1
t:=0
t:=0
t:=1


b1:=T
b1=F
b1=T
b1:=F

Semantics

The behaviour of P is defined by the (reachable subset of) the global transition system

$$T_P = (A, S, \Delta, s_0)$$

where

- $A = A_1 \cup \ldots \cup A_n$
  (A partitioned into visible and invisible actions),
- $S = S_1 \times \ldots \times S_n$
  ($s(i)$ denotes the ith component of $s \in S$),
- $s_0 = (s_{01}, \ldots, s_{0n})$,
- $(s, a, s') \in \Delta$ iff for every $1 \leq i \leq n$
  - $a \in A_i \implies (s(i), a, s'(i)) \in \Delta_i$, and
  - $a \notin A_i \implies s(i) = s'(i)$.

Reducing the model checking problem

The model checking problem for a program $P = (T_1, \ldots, T_n)$ can be reduced to (several instances of) the following problems:

The forbidden trace problem (FT)

Given: Program P, action a.

To decide: Does T_P exhibit a forbidden trace, i.e., a trace $a_0 a_1 a_2 \ldots a_n \in A^*$ such that $a_n = a$?

The forbidden infinite trace problem (FIT)

Given: Program P, action a.

To decide: Does T_P exhibit a forbidden infinite trace, i.e., an infinite trace $a_0 a_1 a_2 \ldots \in A^\omega$ such that $a_i = a$ for infinitely many $i \geq 0$?

The livelock problem (L)

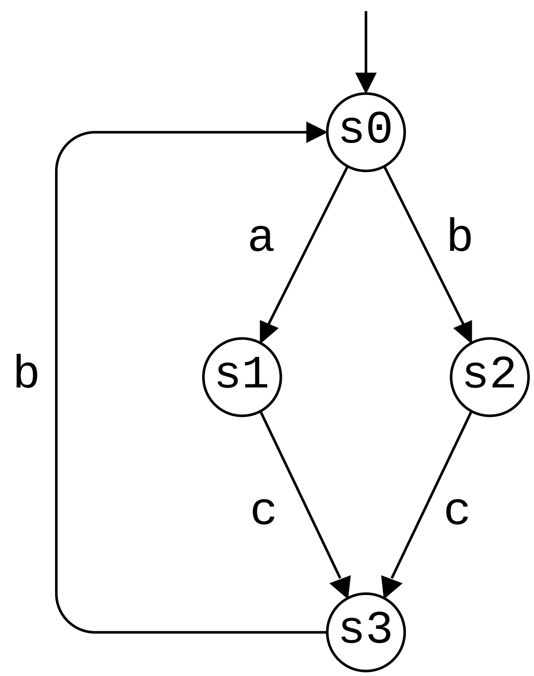
Given: Program P, action a.

To decide: Does T_P exhibit a livelock, i.e., an infinite trace $a_0 a_1 a_2 \ldots \in A^\omega$ such that $a_i = a$ for some i, and a_j is invisible for every $j > i$?

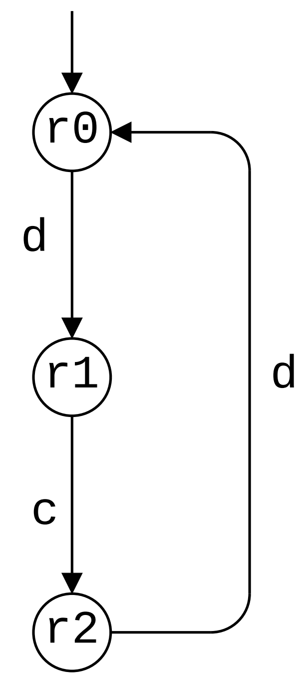
A first analysis

- Complexity of FT, FIT and L: PSPACE-complete.
- Standard solution: compute the global transition system T_P, and use well-known graph algorithms. Time and space complexity $O(|T_P|)$, but in practice often $O(|T_P|^2)$.
- Problem: exponential in the size of P for very easy instances, e.g. for completely independent processes.
- Our solution: work on the unfolding of the system.
  - Compact “proof objects”, exponentially smaller than T_P in favourable cases.
  - “Proof objects” of size $O(|T_P|)$ for FT, and of size $O(|T_P|^2)$ for FIT and L.
  - In this talk: only FT and FIT (L more technical).

Running example

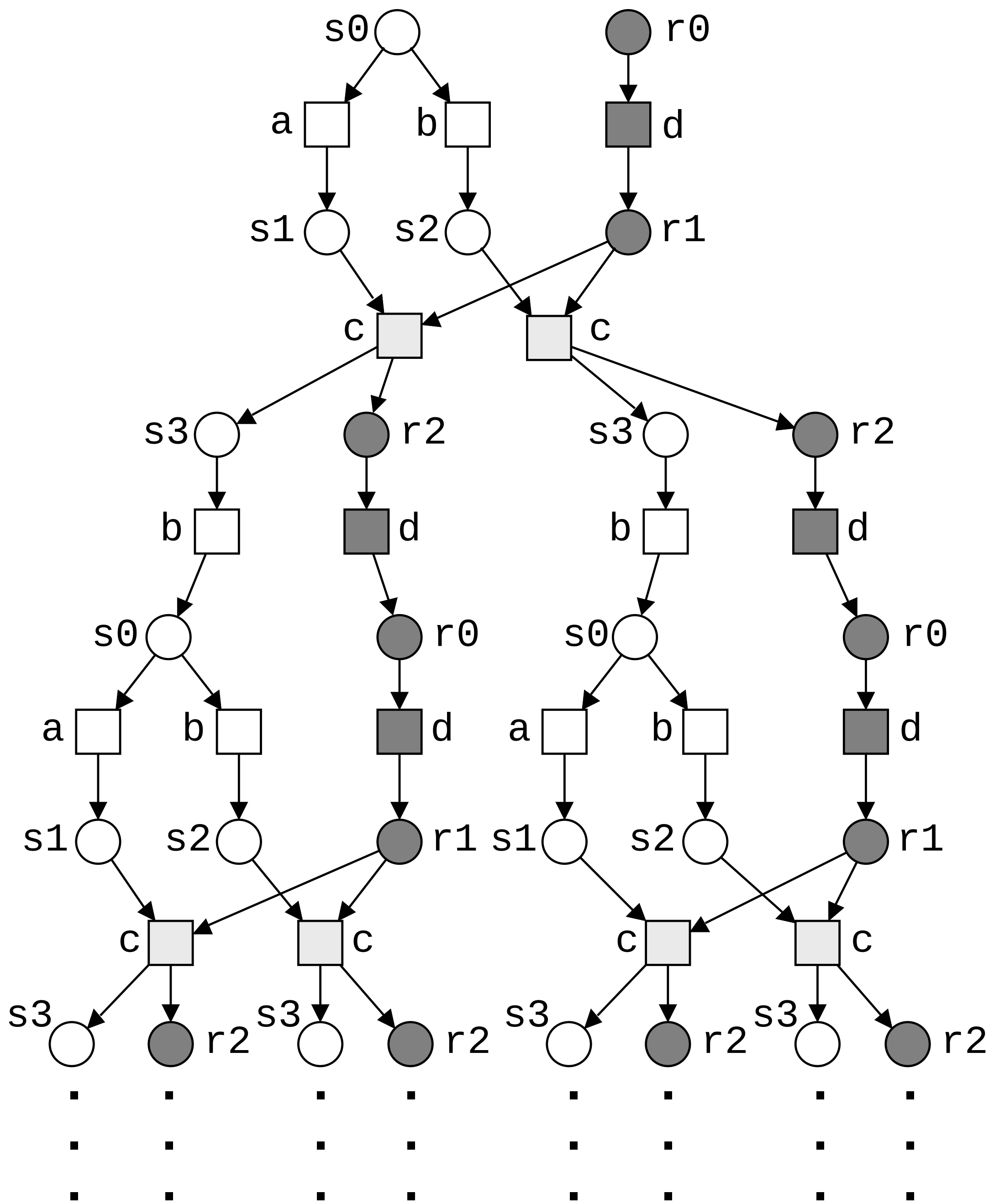


T1

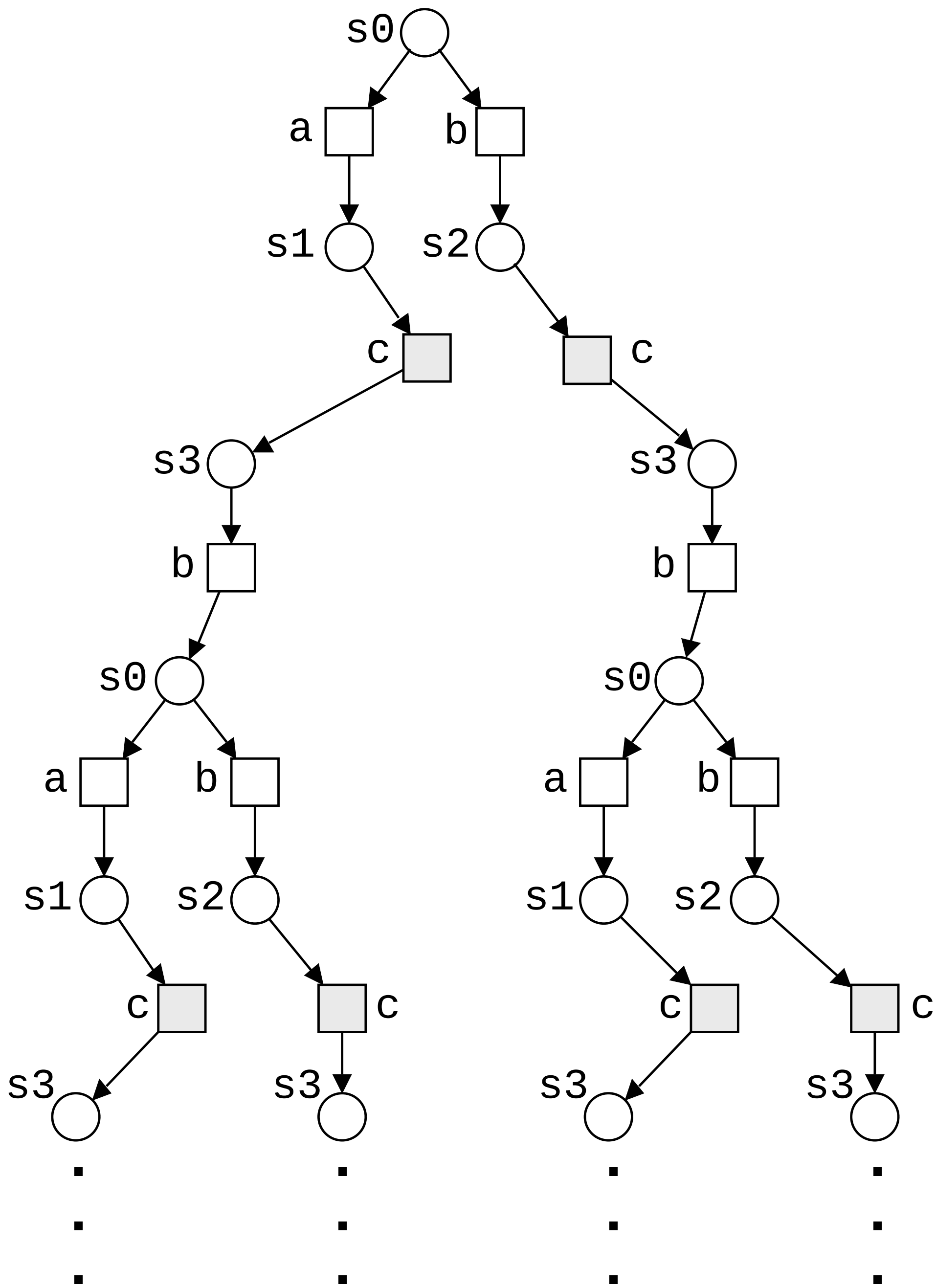


T2

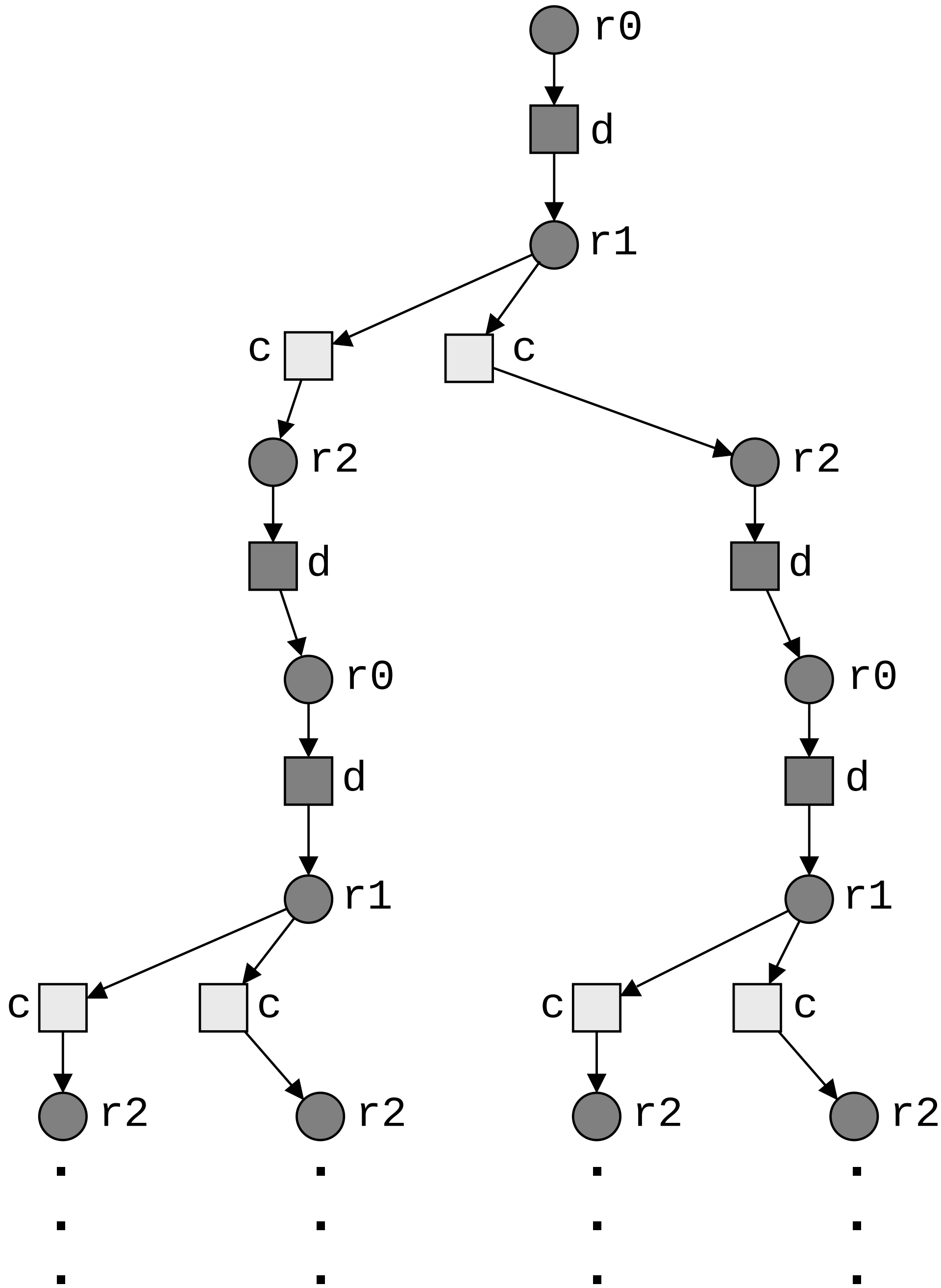
The unfolding



First tree

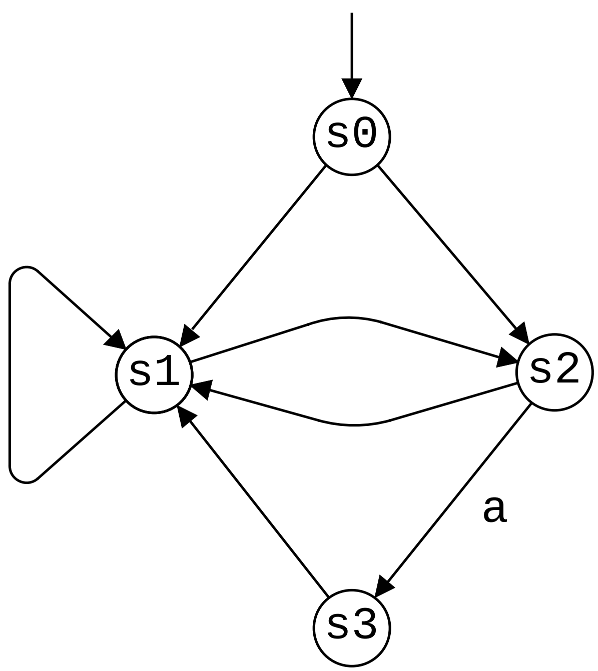


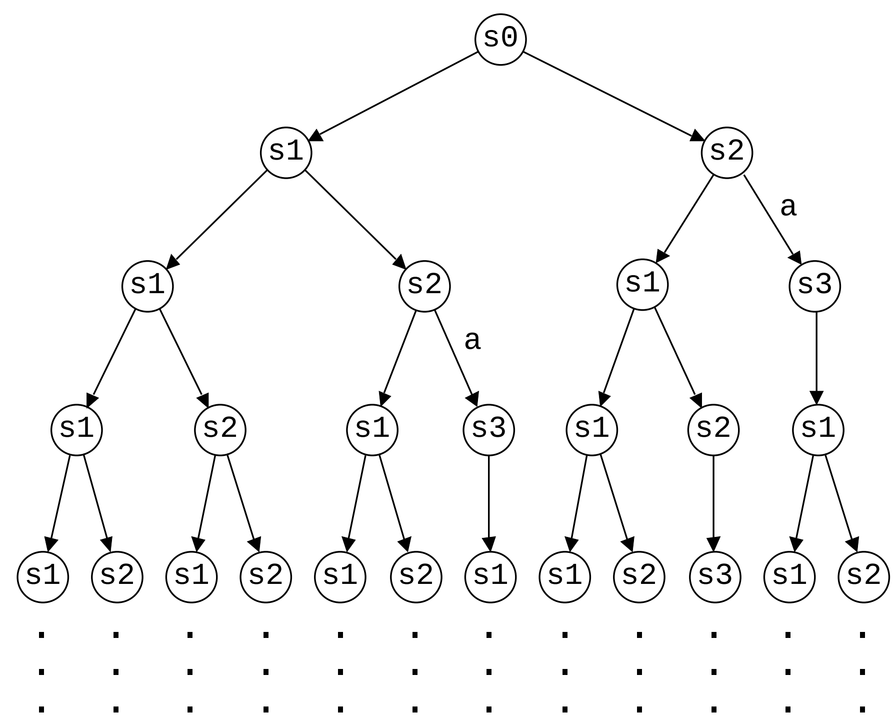
Second tree



An example with $n = 1$

In this case $P = T_1 = T_P$.





Solving FT for $n = 1$

Proof tree: Prefix of the unfolding of P.

A node **n** is a terminal if

(I) it is reached by an event labelled by a, or

(II) the proof tree constructed so far contains a node **n**$'$ labelled by the same state as **n**.

A tableau is a (minimal) proof tree such that all leaves are terminals.

A terminal **n** is successful if it is of type I.

A proof tree is successful if it has at least one successful terminal.

Theorem: (P, a) exhibits a forbidden trace iff (P, a) has a successful proof tree.

Generalization to $n \geq 1$. First attempt

Problem: Terminals are local states, but a terminal's definition must refer to global states.

Idea [McMillan 92, 95]: Associate to each node $\mathbf{n}$ of the unfolding a global state $GS(\mathbf{n})$ as follows:

- let $Hist(\mathbf{n})$ be the "history" of $\mathbf{n}$;
- let $GS(\mathbf{n})$ be the result of "executing" $Hist(\mathbf{n})$.

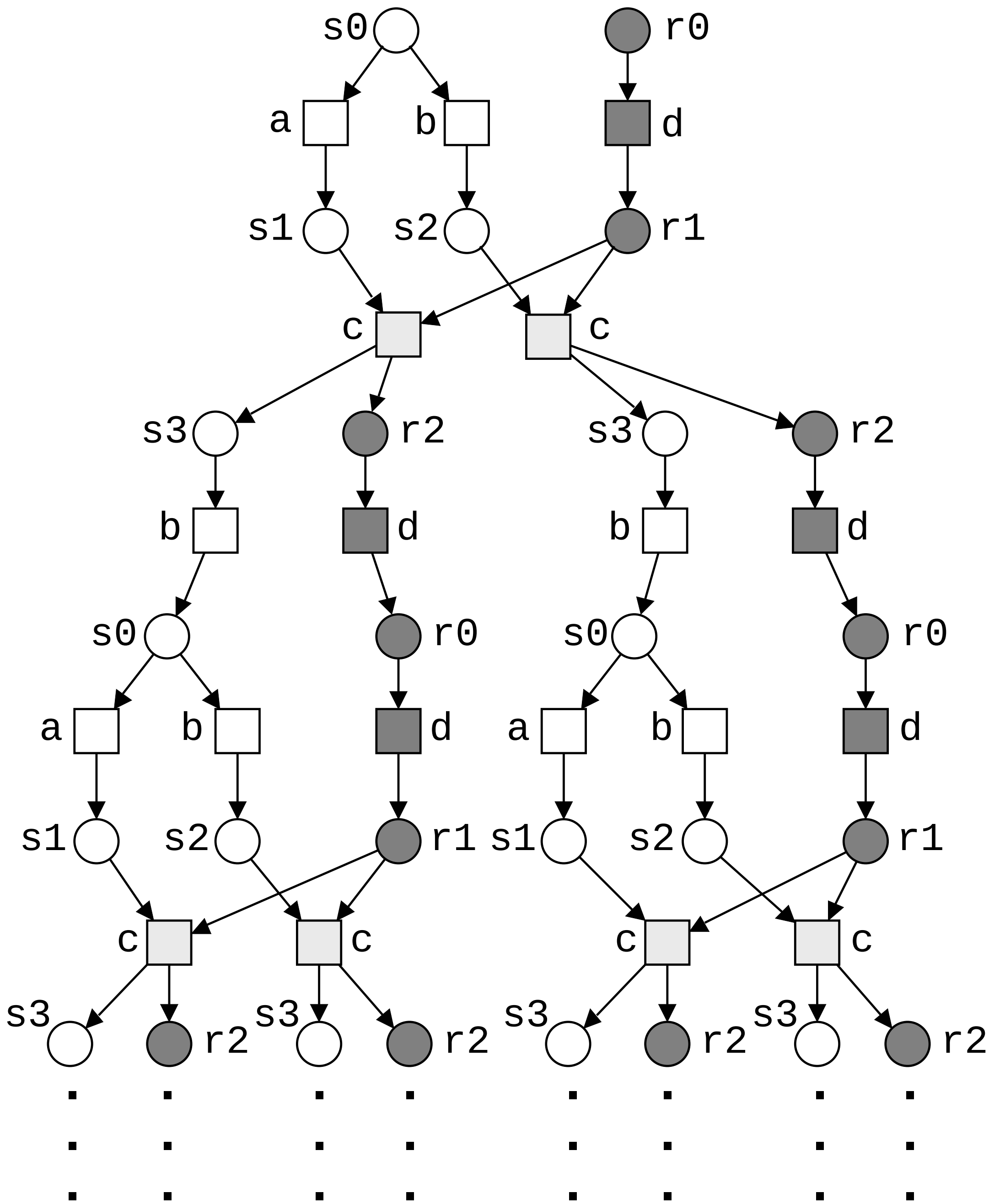
New definition of terminal:

A node $\mathbf{n}$ is a terminal if

(I) it is reached by an event labelled by a, or

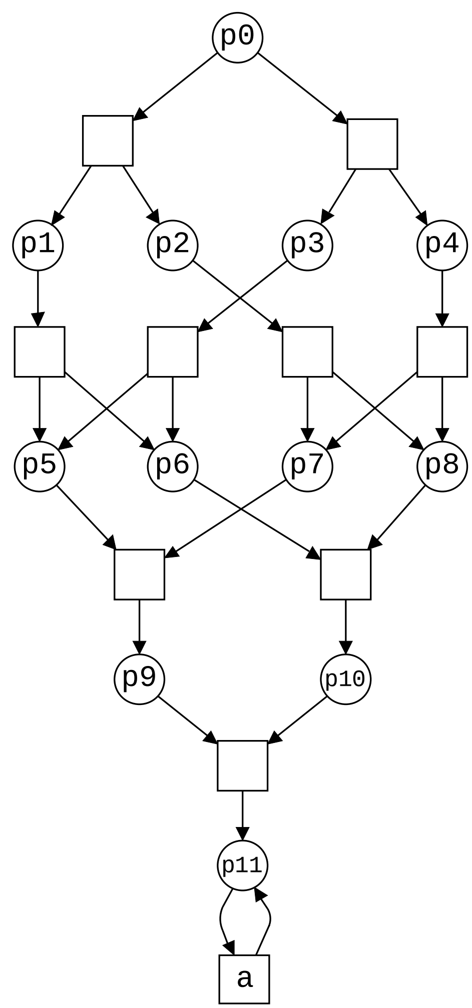
(II) the proof tree constructed so far contains a node $\mathbf{n}'$ such that $GS(\mathbf{n}) = GS(\mathbf{n}')$.

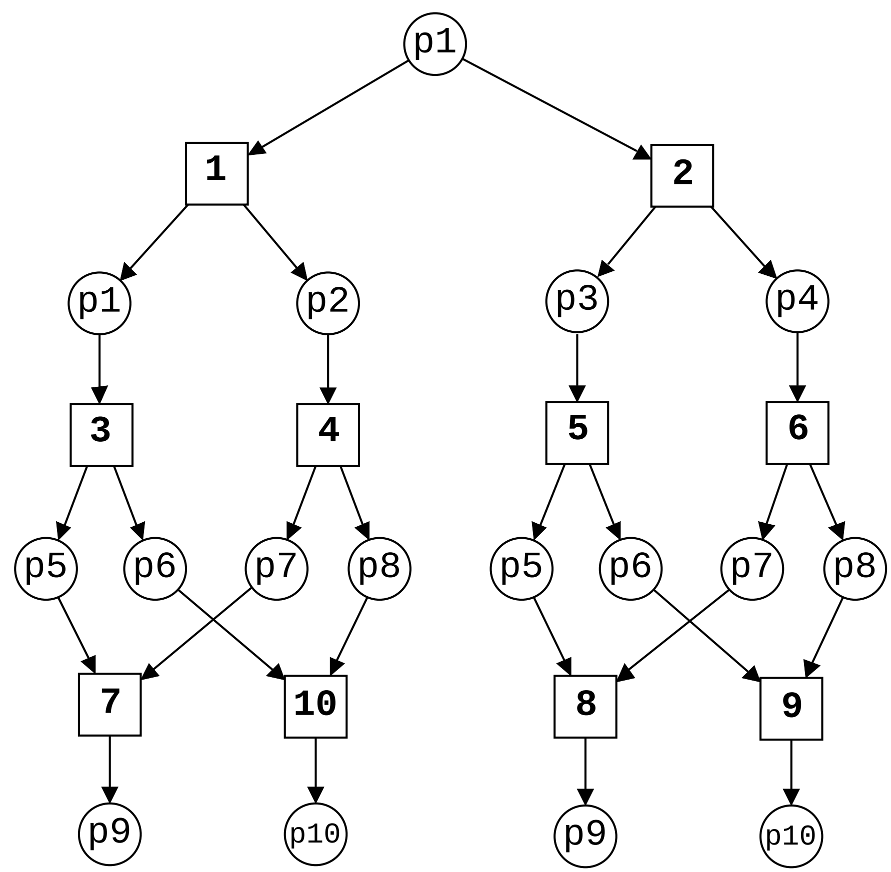
A terminal $\mathbf{n}$ is successful if it is of type I.



s0
r0
a
b
d
s1
s2
r1
c
c
s3
r2
s3
r2
b
d
b
d
s0
r0
s0
r0
a
b
d
a
b
d
s1
s2
r1
s1
s2
r1
c
c
c
c
s3
r2
s3
r2
s3
r2
s3
r2

The attempt fails!





Adequate orders

$GS(\mathbf{n}) = GS(\mathbf{n}')$ too weak for a terminal

An order $\preceq$ on histories is adequate if it

- is well-founded,
- preserves causality: if h prefix of h' then $h \preceq h'$, and
- is preserved by finite extensions: if $h \preceq h'$, then $h \cdot h'' \preceq h' \cdot h''$ for all h''.

We say $\mathbf{n} \preceq \mathbf{n}'$ if $Hist(\mathbf{n}) \preceq Hist(\mathbf{n}')$.

New definition of terminal:

A node $\mathbf{n}$ is a terminal if

(I) it is reached by an a-transition, or

(II) the proof tree constructed so far contains a node $\mathbf{n}' \preceq \mathbf{n}$ such that $GS(\mathbf{n}) = GS(\mathbf{n}')$.

A terminal $\mathbf{n}$ is successful if it is of type I.

Theorem: (P, a) has a forbidden trace iff it has a successful tableau.

Problem: $\mathbf{n}' \preceq \mathbf{n}$ is an additional condition

$\Rightarrow$ less places are terminals

$\Rightarrow$ proof trees can be bigger!

- In [McMillan 92]:
  $h \preceq h'$ if size of h smaller than size of h'.
  Tableaux can be exponentially bigger than T_P

- In [E.,Römer,Vogler 96, E., Römer 99]:
  Total orders $\preceq$.

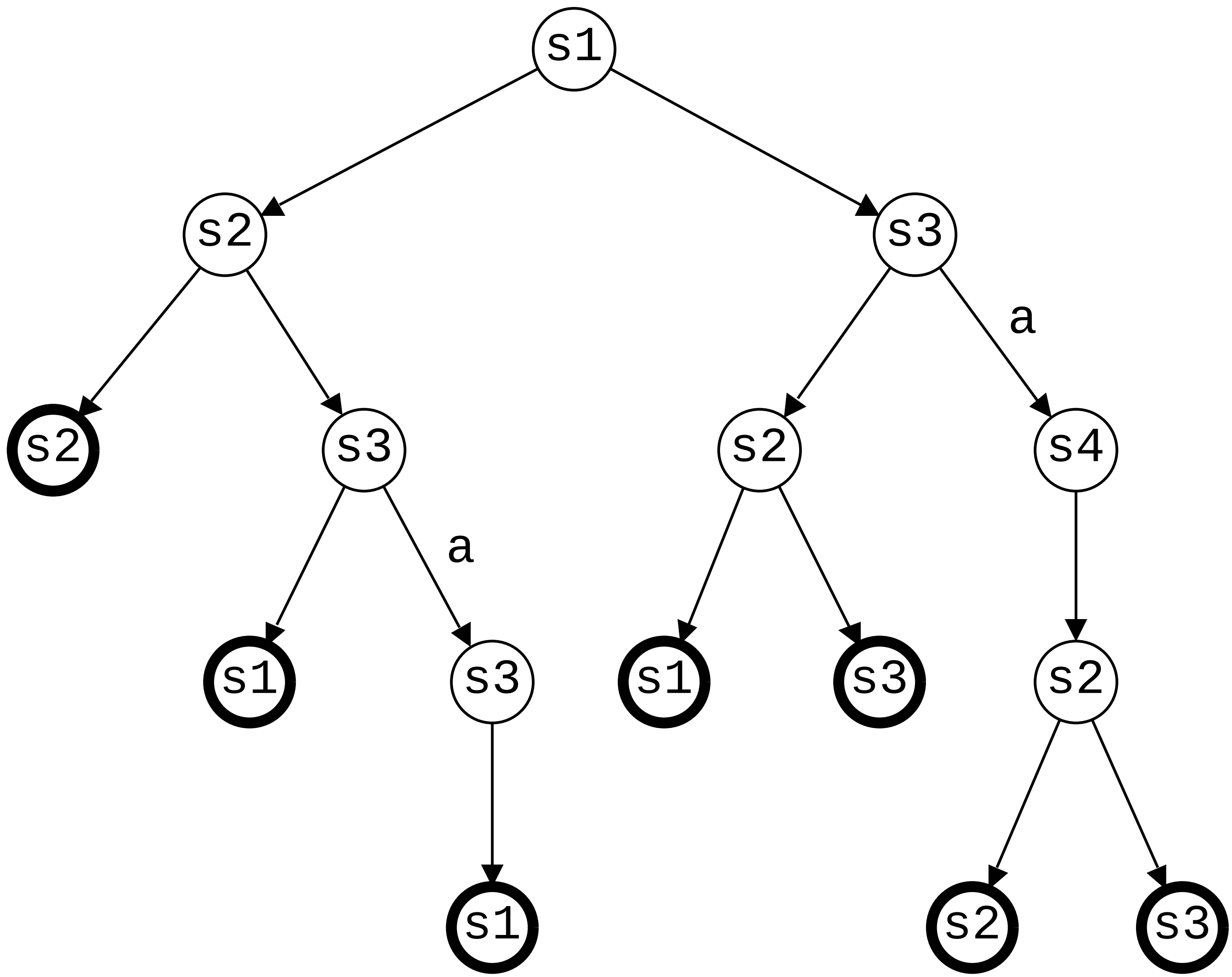
  Theorem: Any tableaux in which the events are added in $\preceq$-order has size $O(T_P)$.

A solution to FIT for $n = 1$

A node $\mathbf{n}$ is a terminal if $Hist(\mathbf{n})$ contains a node $\mathbf{n}'$ labelled with the same state as $\mathbf{n}$.
A terminal $\mathbf{n}$ is successful if $Hist(\mathbf{n}) - Hist(\mathbf{n}')$ (the path from $\mathbf{n}'$ to $\mathbf{n}$) contains some a-labelled transition.

Theorem: (P, a) exhibits a forbidden infinite trace iff (P, a) has a successful tableau.

Problem: tableaux can be exponentially larger than T_P


s1
s2
s3
a
s2
s3
s2
s4
a
s1
s3
s1
s3
s2
s1
s2
s3

A better solution for $n = 1$

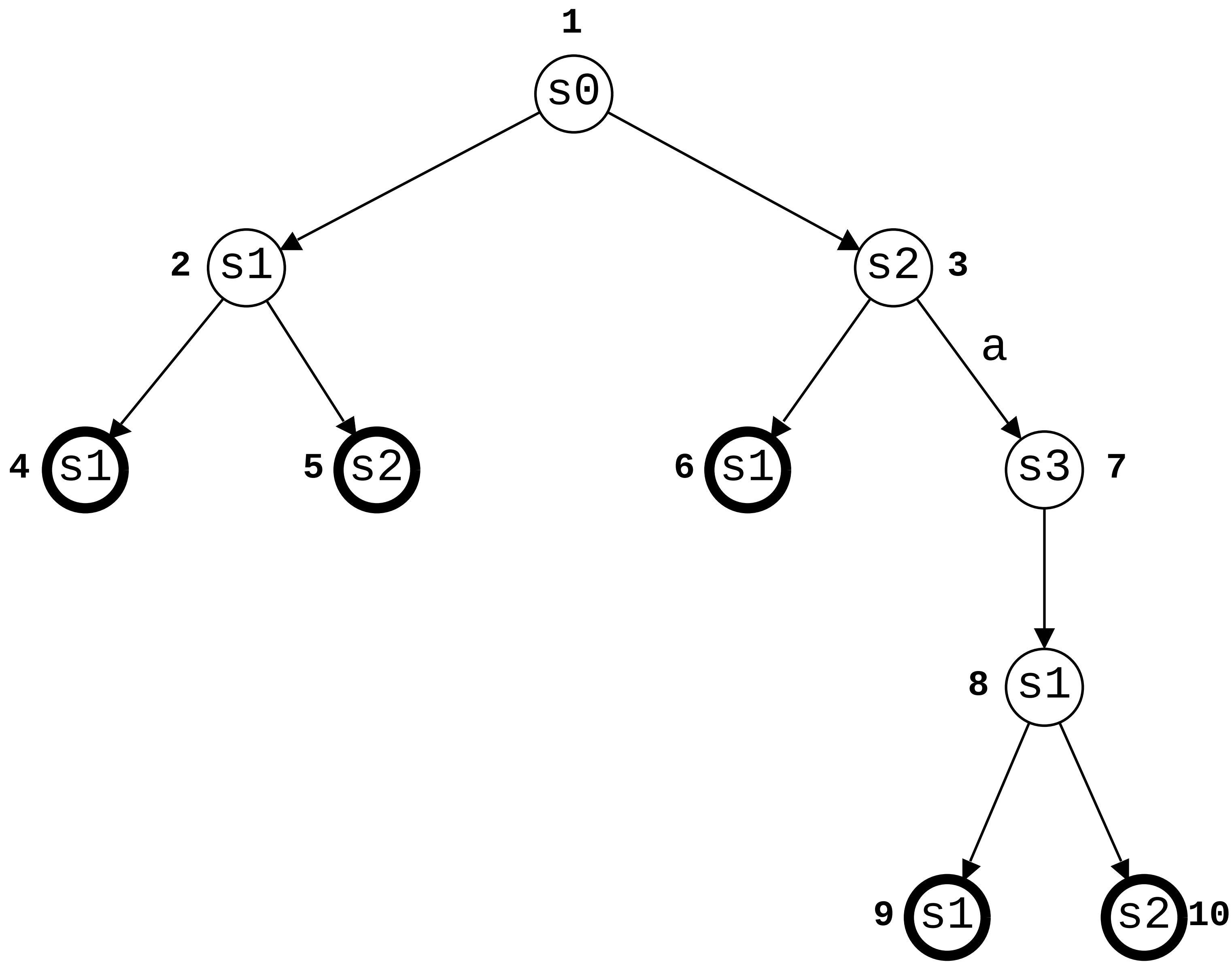
New definition of terminal:

A node $\mathbf{n}$ is a terminal if there is a node $\mathbf{n}'$ labelled with the same state as $\mathbf{n}$ such that

- $\mathbf{n}'$ belongs to $Hist(\mathbf{n})$, or
- $\mathbf{n}'$ does not belong to $Hist(\mathbf{n})$, and $Hist(\mathbf{n}')$ contains at least as many a-labelled transitions as $Hist(\mathbf{n})$.

A terminal $\mathbf{n}$ is successful if $\mathbf{n}'$ belongs to $Hist(\mathbf{n})$ and $Hist(\mathbf{n}) \setminus Hist(\mathbf{n}')$ contains some a-labelled transition.

Theorem: (P, a) has a forbidden infinite trace iff it has a successful tableau. The size of any tableau in which events are added in $\preceq$-order is $O(|T_P|^2)$.


1
s0
2
s1
s2
3
a
4
s1
5
s2
6
s1
s3
7
8
s1
9
s1
s2
10

Generalization to $n > 1$

Definition of terminal:

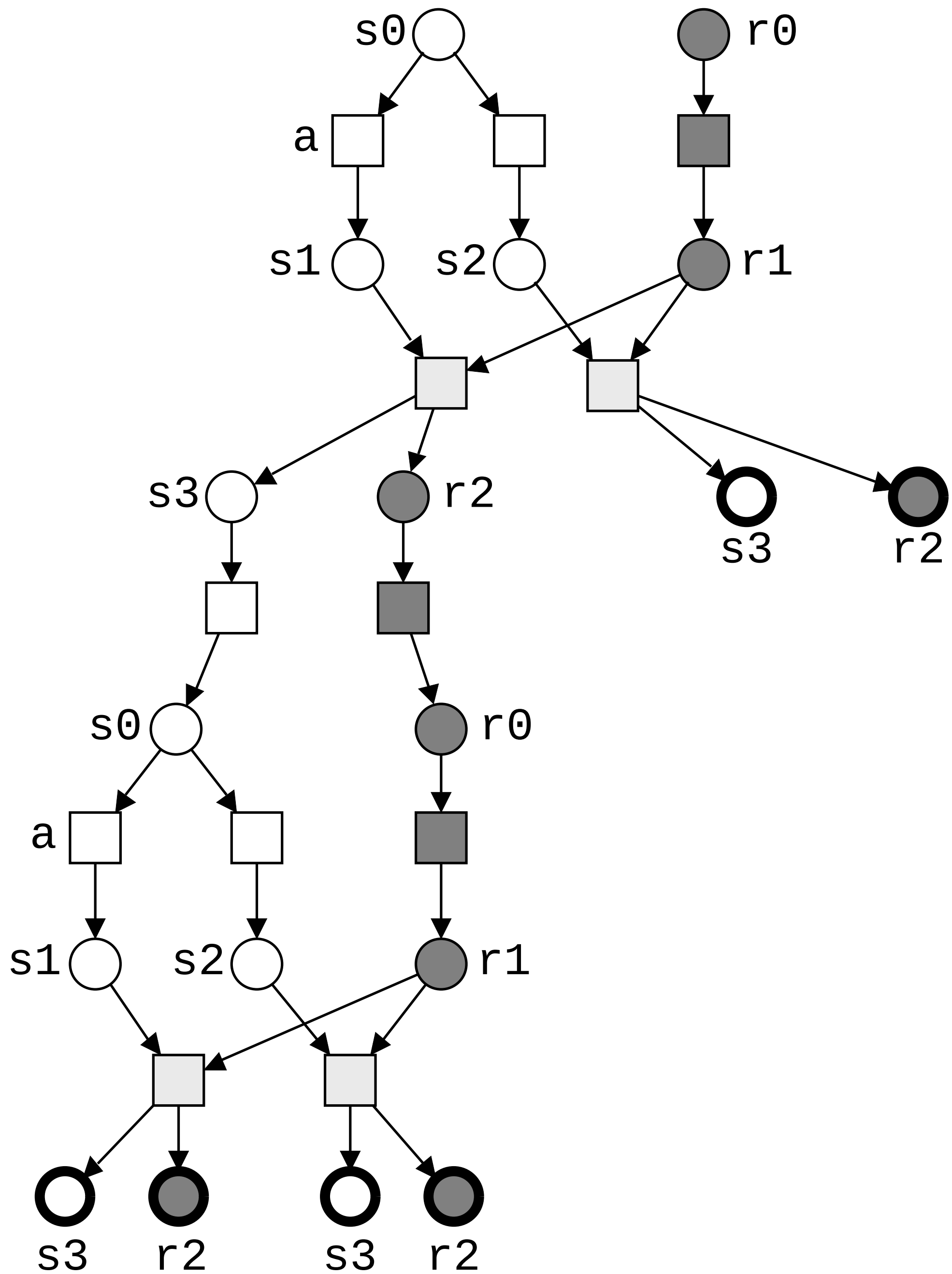
A node $\mathbf{n}$ is a terminal if there is a node $\mathbf{n}' \preceq n$ such that $GS(\mathbf{n}') = GS(\mathbf{n})$, and

- $\mathbf{n}'$ belongs to $Hist(\mathbf{n})$, or
- $\mathbf{n}'$ does not belong to $Hist(\mathbf{n})$, and $Hist(\mathbf{n}')$ contains at least as many a-labelled transitions as $Hist(\mathbf{n})$.

A terminal $\mathbf{n}$ is successful if $\mathbf{n}'$ belongs to $Hist(\mathbf{n})$ and $Hist(\mathbf{n}) \setminus Hist(\mathbf{n}')$ contains some a-labelled transition.

Theorem: (T_P, a) has a forbidden infinite trace iff it has a successful tableau.
With the adequate orders of [E.,Römer,Vogler 96] and [E., Römer 99] the size of any tableau is at most $O(|T_P|^2)$.

A successful tableau



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

CY : Cycler (Milner)

DPD: Philosophers with deadlock avoidance

SR: Slotted ring protocol

EL: Elevator

PC: Production cell

CP: Concurrent Pushers (Heimer)

DME: STG specification of a circuit for distributed mutual exclusion

RW: Readers and writers (Hellwagner)

Unfoldings vs. BDDs

Conceptual similarities and differences:

- Both techniques compress the state space
- BDDs exploit regularity
  Unfoldings exploit concurrency

Consequences:

+ Robustness: Unfoldings less sensitive to changes in the system

+− Compression: Prefix smaller for loosely coupled systems, BDDs smaller for tightly coupled systems

- 100 random tables with right-handed, left-handed, and ambidextrous philosophers
- BDD for the set of reachable states (Petrify)

Nr. of	BDD size				
phil.	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	Prefix size				
phil.	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

Checking deadlock-freedom with BDDs

- 100 random tables with right-handed, left-handed, and ambidextrous philosophers
- SMV on a SUN Ultra 60, 2 processors, 640 MB

Nr of	Time in seconds				
phil.	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-handed, left-handed, and ambidextrous-philosophers
- PEP + stable models on a SUN Ultra 60, 2 processors, 640 MB

Nr. of	Time in seconds				
phil.	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

Unfoldings vs. stubborn sets

Conceptual similarities and differences:

- Both techniques exploit concurrency
- Stubborn sets discard information
  Unfoldings compress information
- Stubborn sets are conservative: small overhead is guaranteed, at the price of a suboptimal reduction
  Unfoldings "take risks": large overhead is possible, but optimal compression

Consequences:

+ No loss of information → All reachability properties checkable on the same prefix

+ Causality information available (ex. alarm patterns)

− Larger overheads for tightly coupled systems

Checking deadlock-fredom with stubborn sets and unfoldings

- 1 left-handed, 1 right-handed, and $(n-2)$ ambidextrous philosophers
- SUN Ultra 60, 2 processors, 640 MB

Nr. of	Time in seconds	
phil.	PROD	PEP + smodels
10	18	0.04
12	69	0.04
14	834	0.04
16	5003	0.04
18	29257	0.06

Tentative rules of thumb

- BDDs more suitable for very regular systems
- Stubborn sets and unfoldings more suitable for irregular but concurrent systems
  - Stubborn sets more suitable for systems with little concurrency
  - Unfoldings more suitable for highly concurrent systems