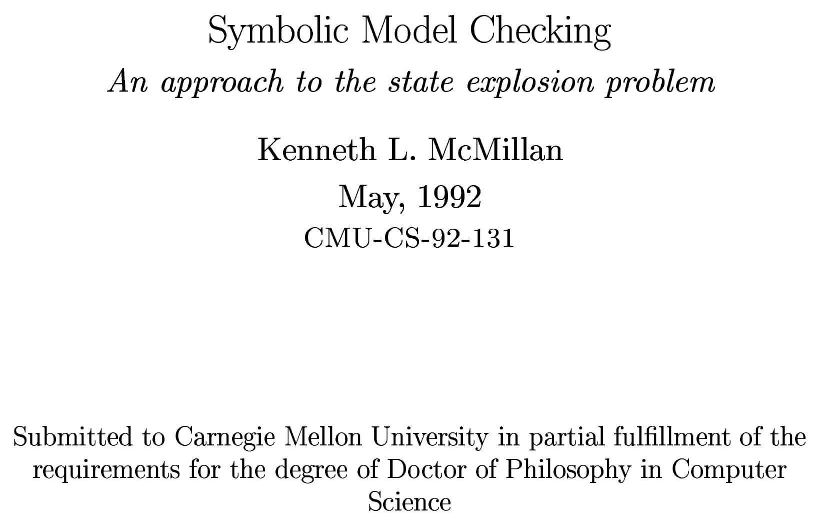
Kapitel 6: Model Checking durch Auffalten



Symbolic Model Checking

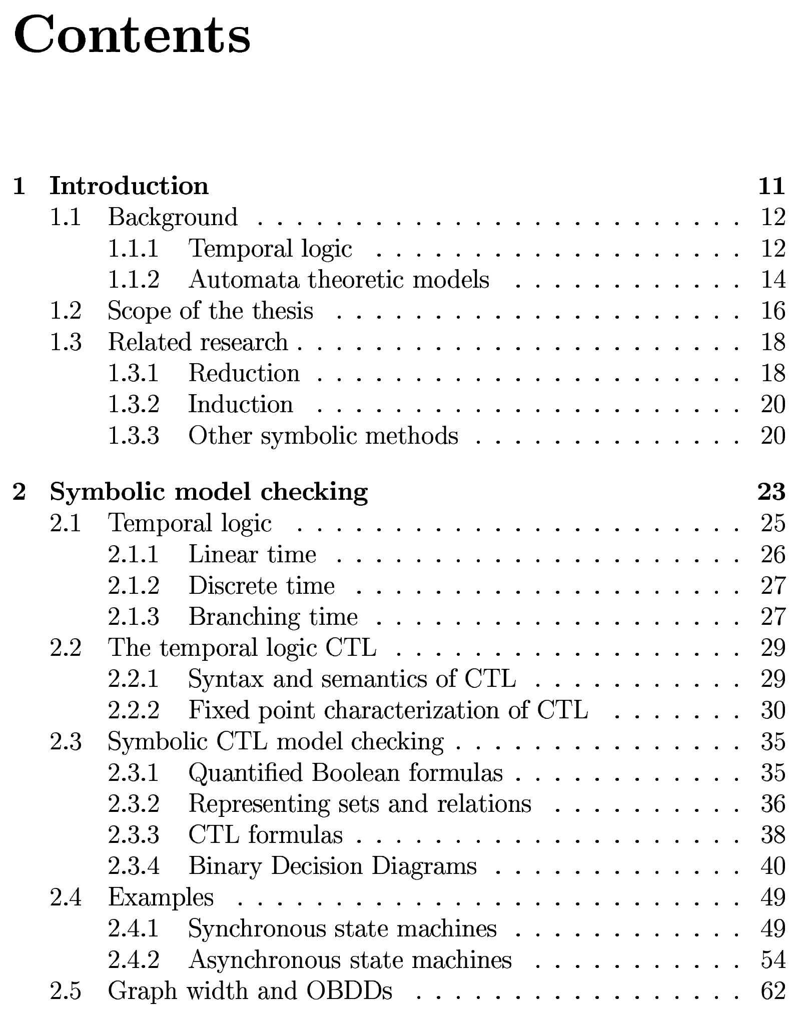
An approach to the state explosion problem

Kenneth L. McMillan

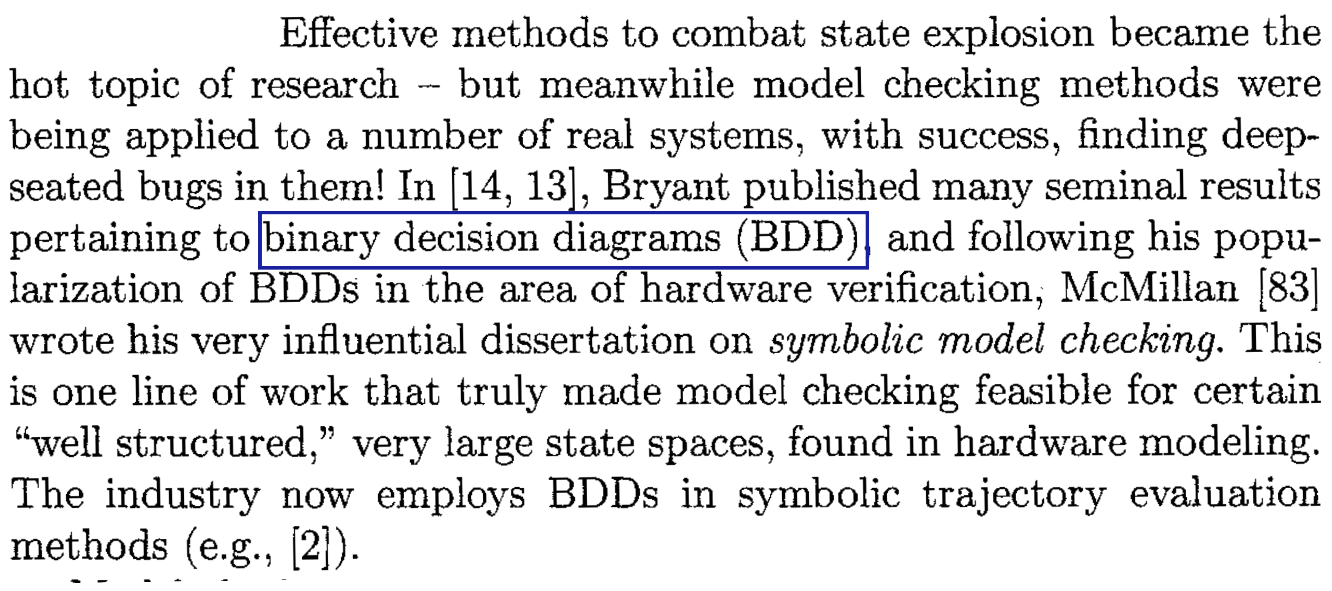
May, 1992

CMU-CS-92-131

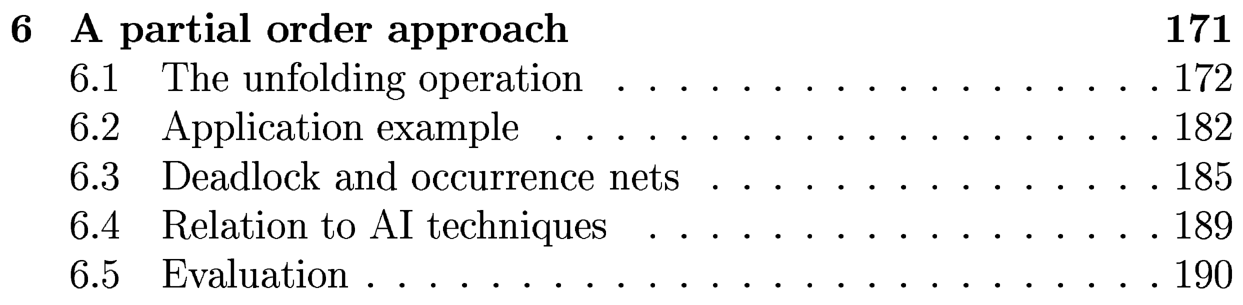
Submitted to Carnegie Mellon University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science



Contents



Effective methods to combat state explosion became the hot topic of research – but meanwhile model checking methods were being applied to a number of real systems, with success, finding deep-seated bugs in them! In [14, 13], Bryant published many seminal results pertaining to binary decision diagrams (BDD), and following his popularization of BDDs in the area of hardware verification, McMillan [83] wrote his very influential dissertation on *symbolic model checking*. This is one line of work that truly made model checking feasible for certain "well structured," very large state spaces, found in hardware modeling. The industry now employs BDDs in symbolic trajectory evaluation methods (e.g., [2]).



K. L. McMillan: A Technique of a State Space Search Based on Unfolding

Formal Methods in System Design 6 (1995) 1, 45-65

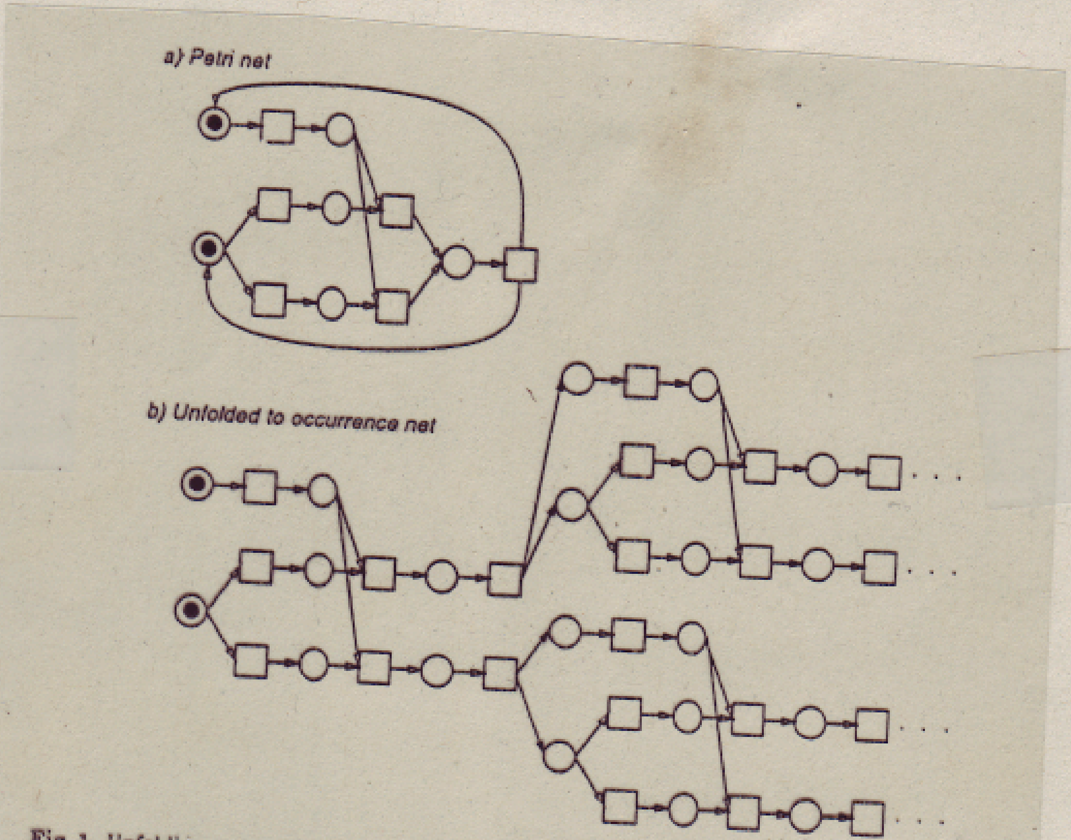
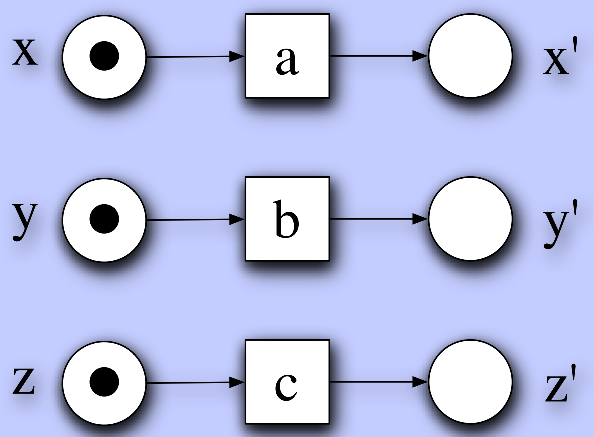


Fig. 1. Unfolding example.

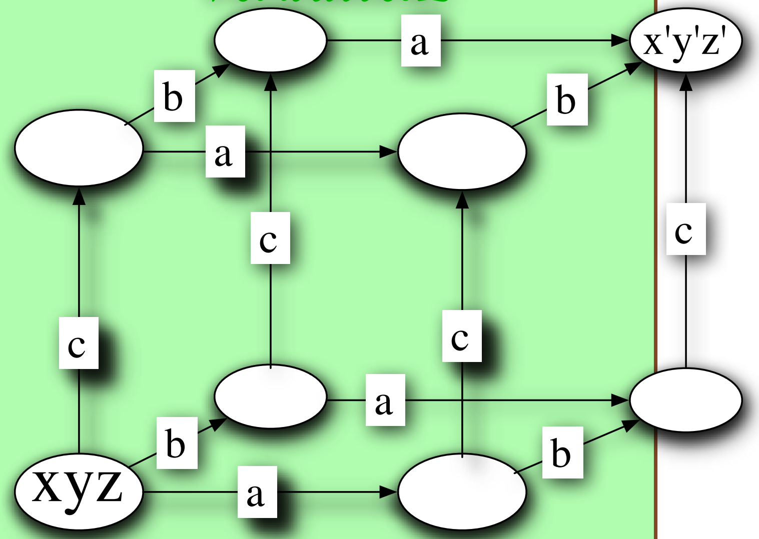
Modell



Petrinetz

....

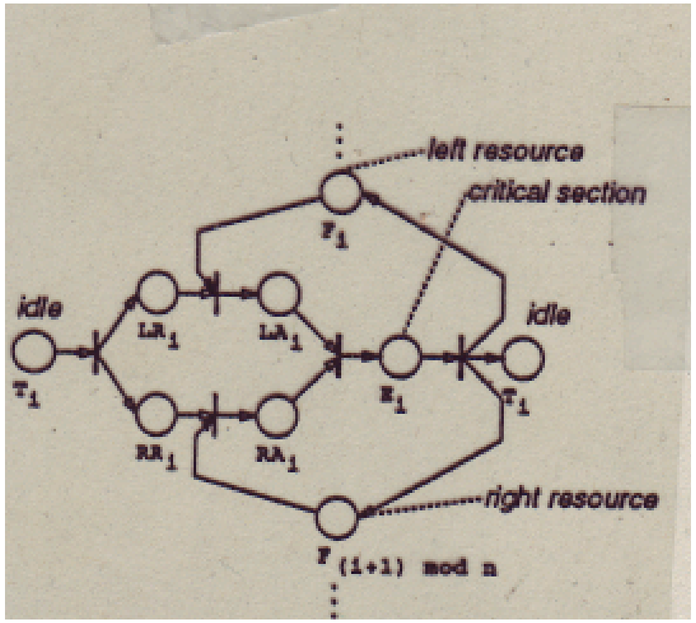
Verhalten

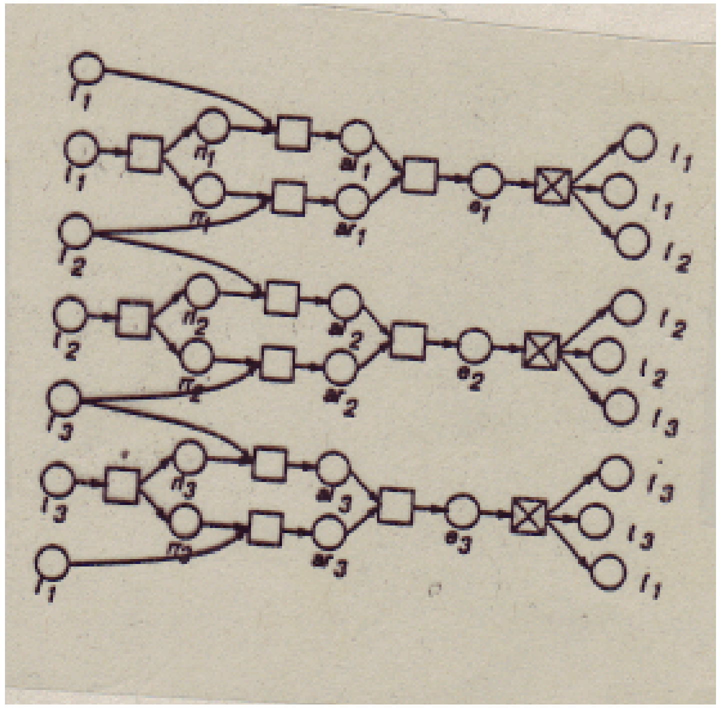


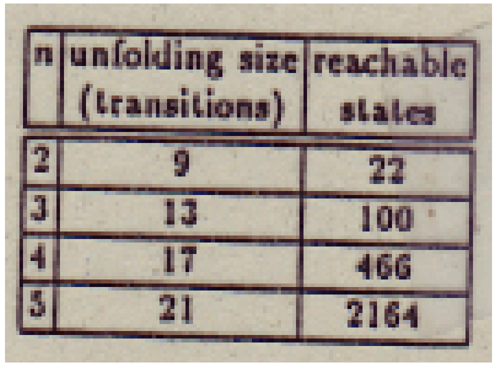
Erreichbarkeitsgraph
Markierungsgraph

....

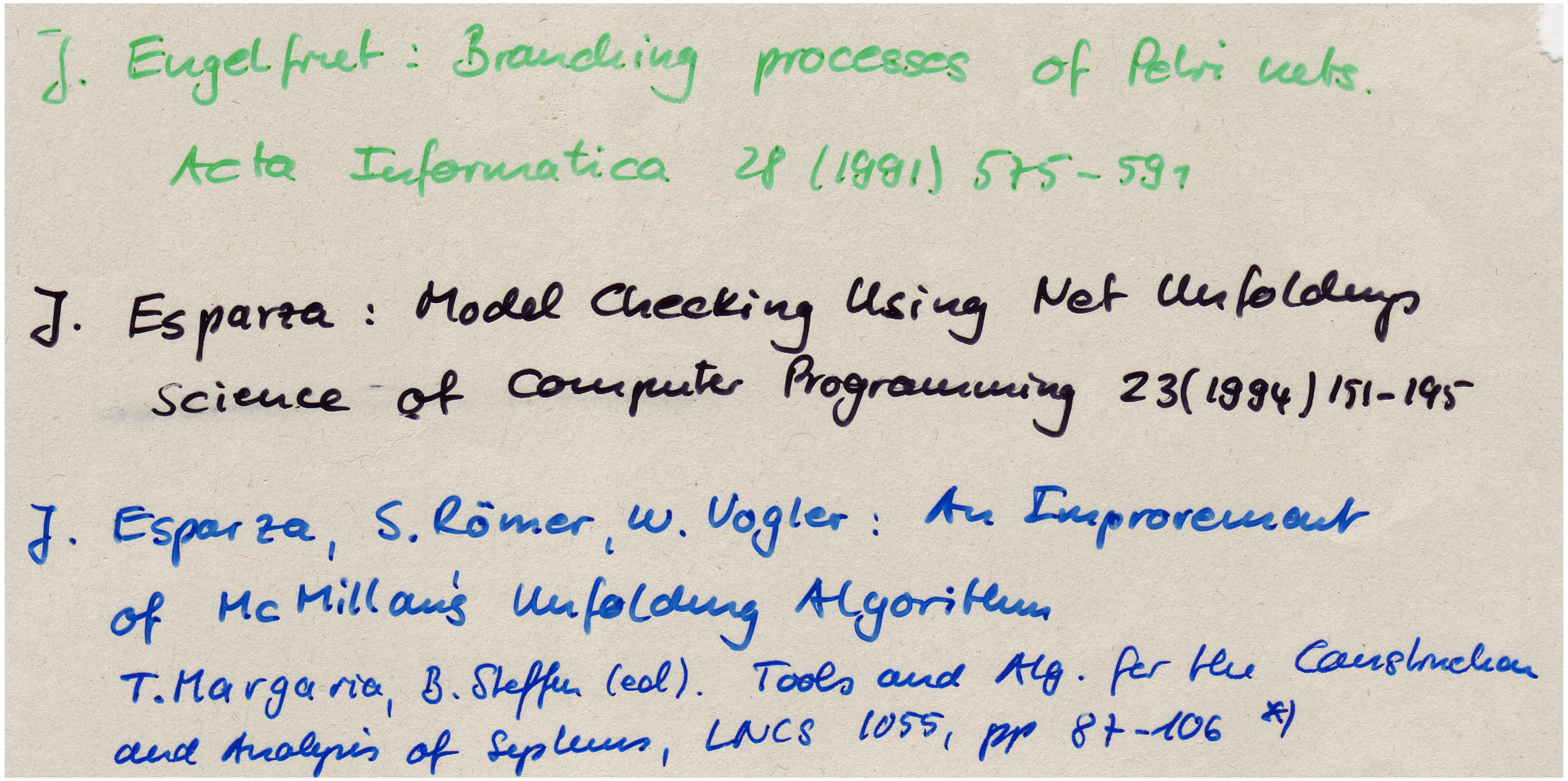
Zustandsexplosion







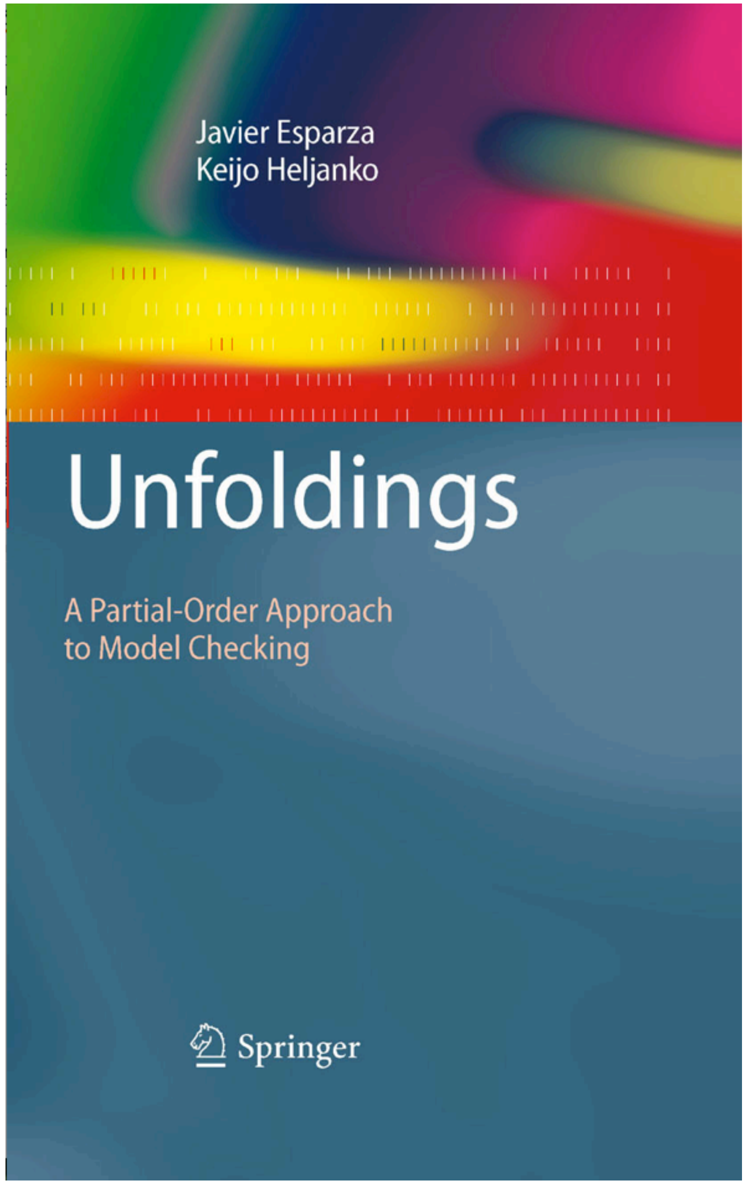
n	unfolding size (transitions)	reachable states
2	9	22
3	13	100
4	17	466
5	21	2164



J. Engelfriet: Branching processes of Petri nets.
Acta Informatica 28 (1991) 575-591

J. Esparza: Model Checking Using Net Unfoldings
Science of Computer Programming 23(1994) 151-195

J. Esparza, S. Römer, W. Vogler: An Improvement of McMillan's Unfolding Algorithm
T. Margaria, B. Steffen (ed). Tools and Alg. for the Construction and Analysis of Systems, LNCS 1055, pp 87-106 *)



free download:
http://freebooksource.com/?p=7048

kaufen &info:
http://www.springer.com/computer/foundations/book/978-3-540-77425-9

2

Transition Systems and Products

In this chapter we introduce transition systems as a formal model of sequential systems, and synchronous products of transition systems as a model of concurrent systems.

2.1 Transition Systems

A *transition system* is a tuple $\mathcal{A} = \langle S, T, \alpha, \beta, is \rangle$, where

- S is a set of *states*,
- T is a set of *transitions*,
- $\alpha: T \to S$ associates with each transition its *source* state,
- $\beta: T \to S$ associates with each transition its *target* state, and
- $is \in S$ is the *initial state*.

Example 2.1. Figure 2.1 shows a transition system $\mathcal{A} = \langle S, T, \alpha, \beta, is \rangle$ where $S = \{s_1, s_2, s_3, s_4\}$, $T = \{t_1, t_2, t_3, t_4, t_5\}$, and $is = s_1$. We have for instance $\alpha(t_1) = s_1$ and $\beta(t_1) = s_2$.

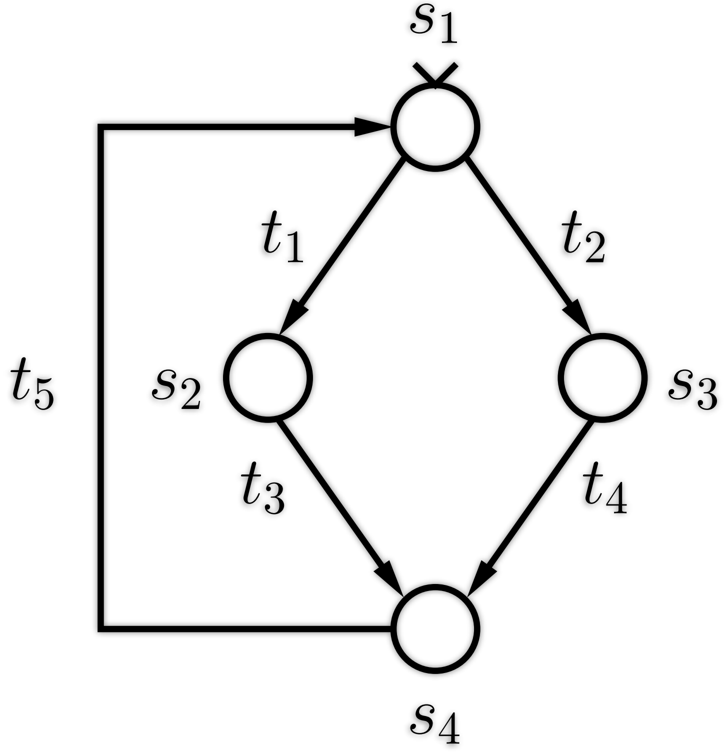


Fig. 2.1. A transition system

We call a finite or infinite sequence of transitions a *transition word* or just a *word*. Given a transition t, we call the triple $\langle \alpha(t), t, \beta(t) \rangle$ a *step* of $\mathcal{A}$. A state s *enables* a transition t if there is a state s' such that $\langle s, t, s' \rangle$ is a step. A (possibly empty) transition word $t_1 t_2 \ldots t_k$ is a *computation* of $\mathcal{A}$ if there is a sequence $s_0 s_1 \ldots s_k$ of states such that $\langle s_{i-1}, t_i, s_i \rangle$ is a step for every $i \in \{1, \ldots, k\}$;[1] we say that the computation starts at s_0 and leads to s_k. A computation is a *history* if $s_0 = is$, i.e., if it can be executed from the initial state. An infinite word $t_1 t_2 \ldots$ is an *infinite computation* of $\mathcal{A}$ if there is an infinite sequence $s_0, s_1, \ldots$ of states such that $\langle s_{i-1}, t_i, s_i \rangle$ is a step for every $i \geq 1$, and an *infinite history* if moreover $s_0 = is$.

If h is a history leading to a state s and c is a computation that can be executed from s, then hc is also a history. We then say that h *can be extended* by c.

2.2 Products of Transition Systems

Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be transition systems, where $\mathcal{A}_i = \langle S_i, T_i, \alpha_i, \beta_i, is_i \rangle$. A *synchronization constraint* $\mathbf{T}$ is a subset of the set

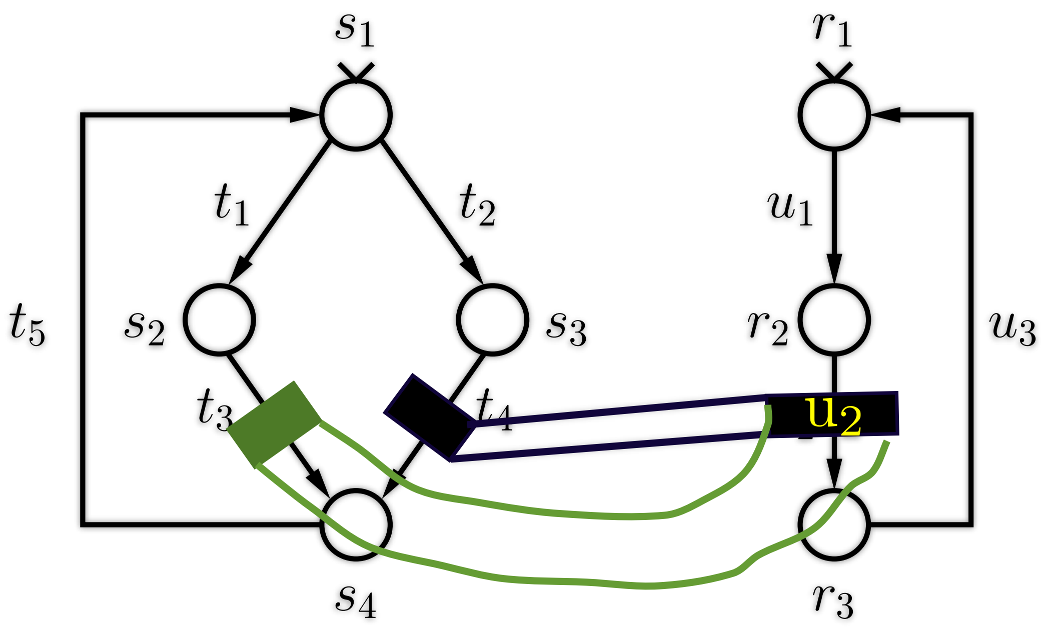
$$(T_1 \cup \{\epsilon\}) \times \cdots \times (T_n \cup \{\epsilon\}) \setminus \{\langle \epsilon, \ldots, \epsilon \rangle\}$$

where ϵ is an special symbol intended to denote inaction (idling). The elements of $\mathbf{T}$ are called *global transitions*. If $\mathbf{t} = \langle t_1, \ldots, t_n \rangle$ and $t_i \neq \epsilon$, then we say that $\mathcal{A}_i$ *participates* in $\mathbf{t}$.[2] The tuple $\mathbf{A} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathbf{T} \rangle$ is called the *product* of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ under $\mathbf{T}$. $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are the *components* of $\mathbf{A}$.

Intuitively, a global transition $\mathbf{t} = \langle t_1, \ldots t_n \rangle$ models a possible move of $\mathcal{A}_1, \ldots, \mathcal{A}_n$. If $t_i = \epsilon$, then $\mathbf{t}$ can occur without $\mathcal{A}_i$ even “noticing”.

[2] This is the reason why $\langle \epsilon, \ldots, \epsilon \rangle$ is excluded from the set of global transitions: at least one component must participate in every global transition.

Example 2.2. Figure 2.2 shows a product of transition systems with two components and seven global transitions. The first component participates in five of them, and the second component in four.



$$\mathbf{T} = \{\langle t_1, \epsilon\rangle, \langle t_2, \epsilon\rangle, \boxed{\langle t_3, u_2\rangle}, \boxed{\langle t_4, u_2\rangle}, \langle t_5, \epsilon\rangle, \langle \epsilon, u_1\rangle, \langle \epsilon, u_3\rangle\}$$

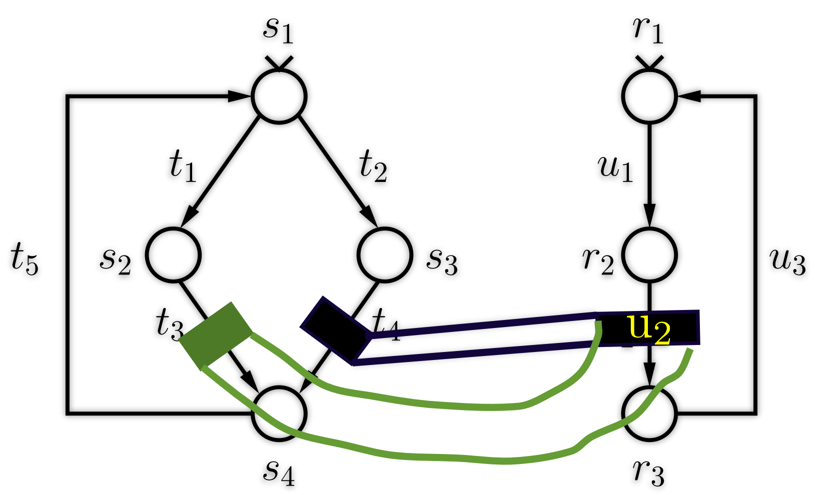
Fig. 2.2. A product of transition systems

A *global state* of $\mathbf{A}$ is a tuple $\mathbf{s} = \langle s_1, \dots, s_n \rangle$, where $s_i \in S_i$ for every $i \in \{1, \dots, n\}$. The *initial global state* is the tuple $\mathbf{is} = \langle is_1, \dots, is_n \rangle$.

A *step* of $\mathbf{A}$ is a triple $\langle \mathbf{s}, \mathbf{t}, \mathbf{s}' \rangle$, where $\mathbf{s} = \langle s_1, \dots, s_n \rangle$ and $\mathbf{s}' = \langle s'_1, \dots, s'_n \rangle$ are global states and $\mathbf{t} = \langle t_1, \dots, t_n \rangle$ is a global transition satisfying the following conditions for all $i \in \{1, \dots, n\}$:

- if $t_i \neq \epsilon$, then $s'_i = \beta_i(t_i)$ and $s_i = \alpha(t_i)$; and
- if $t_i = \epsilon$, then $s'_i = s_i$.

We say that $\mathbf{s}$ *enables* $\mathbf{t}$ if there is a global state $\mathbf{s}'$ such that $\langle \mathbf{s}, \mathbf{t}, \mathbf{s}' \rangle$ is a step.



$$\mathbf{T} = \{\langle t_1, \epsilon\rangle, \langle t_2, \epsilon\rangle, \boxed{\langle t_3, u_2\rangle}, \boxed{\langle t_4, u_2\rangle}, \langle t_5, \epsilon\rangle, \langle \epsilon, u_1\rangle, \langle \epsilon, u_3\rangle\}$$

Fig. 2.2. A product of transition systems

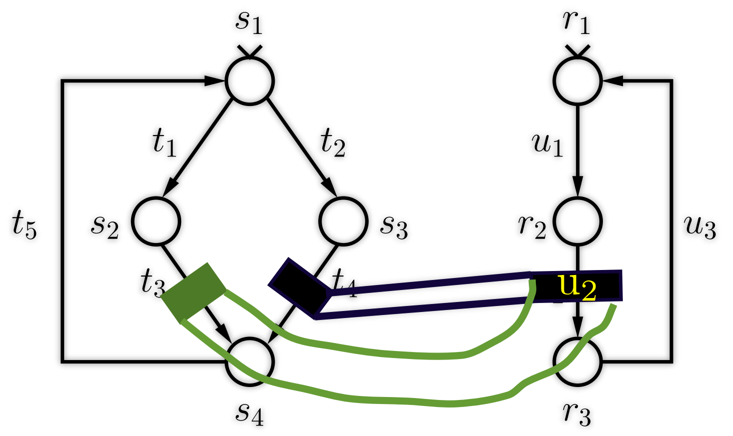
Example 2.3. Consider the product of Fig. 2.2. The initial global state is $\langle s_1, r_1\rangle$. The global transition word $\langle t_1, \epsilon\rangle \langle \epsilon, u_1\rangle \langle t_3, u_2\rangle$ is a global computation, because of the following three steps:

$$\begin{aligned}
&\langle\langle s_1, r_1\rangle, \langle t_1, \epsilon\rangle, \langle s_2, r_1\rangle\rangle, \\
&\langle\langle s_2, r_1\rangle, \langle \epsilon, u_1\rangle, \langle s_2, r_2\rangle\rangle, \text{ and} \\
&\langle\langle s_2, r_2\rangle, \langle t_3, u_2\rangle, \langle s_4, r_3\rangle\rangle.
\end{aligned}$$

The sequence $\langle t_1, \epsilon\rangle \langle t_3, u_1\rangle$ is not a global computation, because $\langle t_3, u_1\rangle$ is not a global transition.

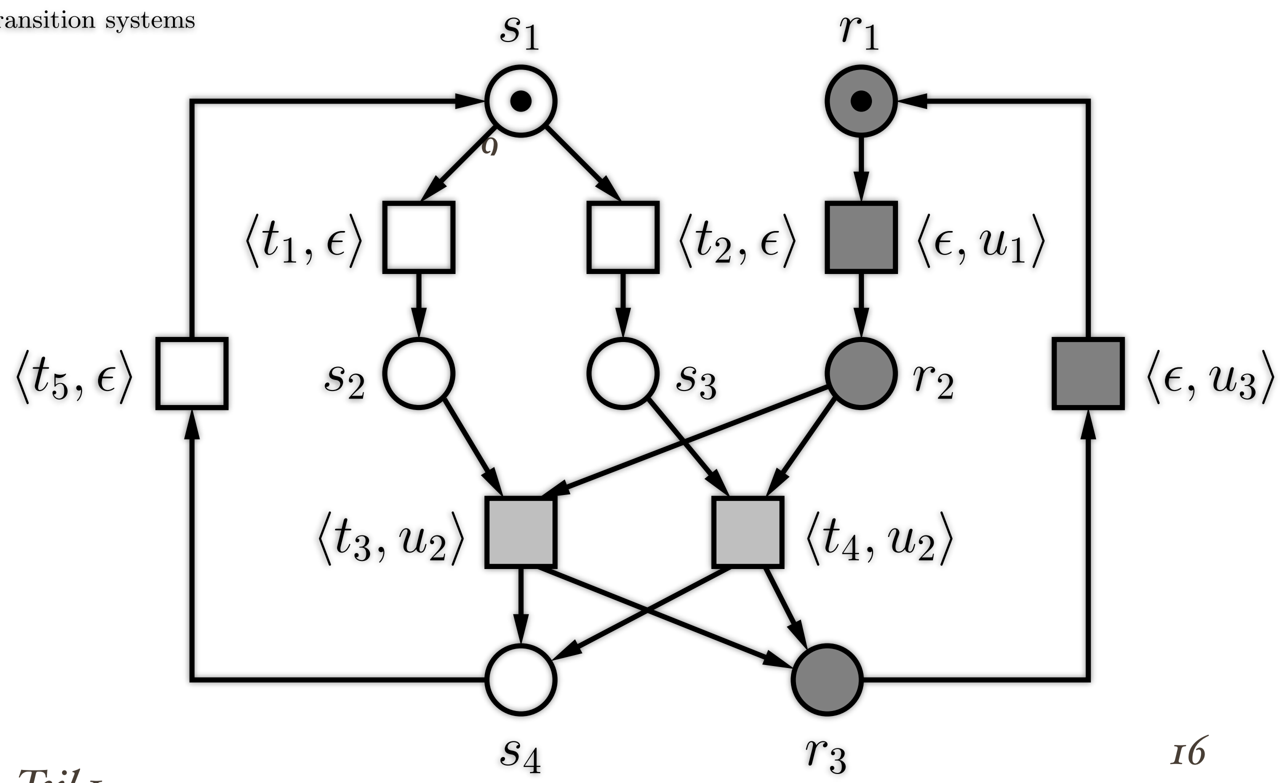
The *Petri net representation* of a product $\mathbf{A} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathbf{T} \rangle$ of transition systems $\mathcal{A}_i = \langle S_i, T_i, \alpha_i, \beta_i, is_i \rangle$ is the Petri net (P, T, F, M_0) given by:

- $P = S_1 \cup \ldots \cup S_n$,[4]
- $T = \mathbf{T}$,
- $F = \{(s, \mathbf{t}) \mid t_i \neq \epsilon$ and $s = \alpha_i(t_i)$ for some $i \in \{1, \ldots, n\}\} \cup$
  $\{(\mathbf{t}, s) \mid t_i \neq \epsilon$ and $s = \beta_i(t_i)$ for some $i \in \{1, \ldots, n\}\}$,
  where t_i denotes the i-th component of $\mathbf{t} \in \mathbf{T}$; and
- $M_0 = \{is_1, \ldots, is_n\}$.



$\mathbf{T} = \{\langle t_1, \epsilon\rangle, \langle t_2, \epsilon\rangle, \boxed{\langle t_3, u_2\rangle}, \boxed{\langle t_4, u_2\rangle}, \langle t_5, \epsilon\rangle, \langle \epsilon, u_1\rangle, \langle \epsilon, u_3\rangle\}$

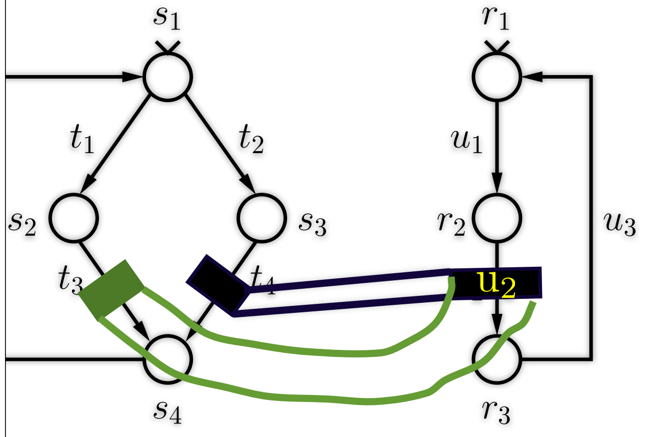
Fig. 2.2. A product of transition systems


s_1
r_1
$\langle t_1, \epsilon\rangle$
$\langle t_2, \epsilon\rangle$
$\langle \epsilon, u_1\rangle$
$\langle t_5, \epsilon\rangle$
s_2
s_3
r_2
$\langle \epsilon, u_3\rangle$
$\langle t_3, u_2\rangle$
$\langle t_4, u_2\rangle$
s_4
r_3

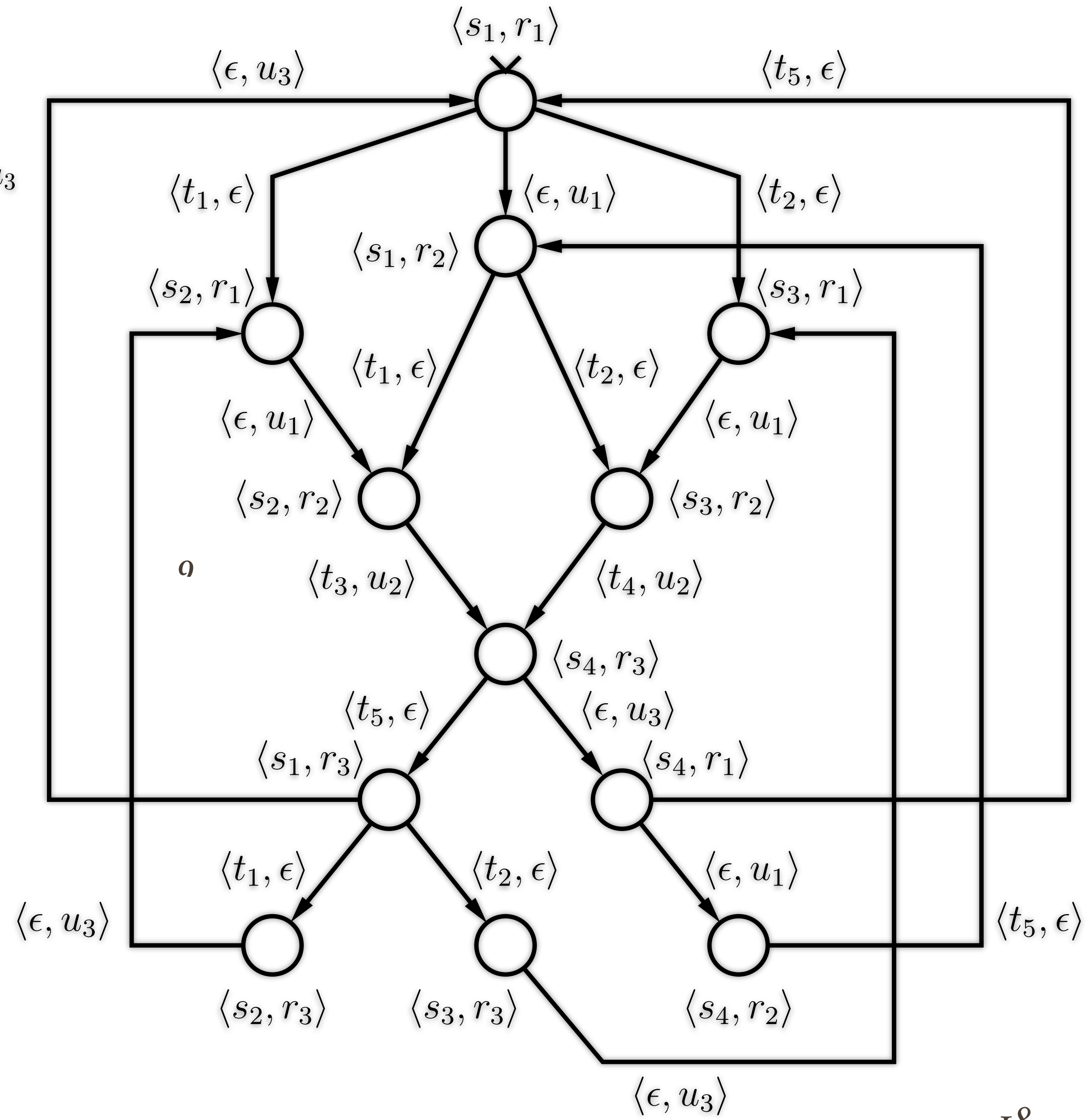
Formally, the *interleaving semantics* of a product $\mathbf{A} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathbf{T} \rangle$ is the transition system $\mathcal{T}_{\mathbf{A}} = \langle S, T, \alpha, \beta, is \rangle$, where

- S is the set of global states of $\mathbf{A}$,
- T is the set of steps $\langle \mathbf{s}, \mathbf{t}, \mathbf{s}' \rangle$ of $\mathbf{A}$,
- for every step $\langle \mathbf{s}, \mathbf{t}, \mathbf{s}' \rangle \in T$: $\alpha(\langle \mathbf{s}, \mathbf{t}, \mathbf{s}' \rangle) = \mathbf{s}$ and $\beta(\langle \mathbf{s}, \mathbf{t}, \mathbf{s}' \rangle) = \mathbf{s}'$; and
- $is = \mathbf{is}$.

Observe that $|S| = \prod_{i=1}^{n} |S_i|$, and so the interleaving semantics of $\mathbf{A}$ can be exponentially larger than $\mathbf{A}$, even if we consider only the states that are reachable from the initial state.



s_1
r_1
t_1
t_2
u_1
s_2
s_3
r_2
u_3
t_3
t_4
u2
s_4
r_3



$\langle s_1, r_1\rangle$
$\langle \epsilon, u_3\rangle$
$\langle t_5, \epsilon\rangle$
$\langle t_1, \epsilon\rangle$
$\langle \epsilon, u_1\rangle$
$\langle t_2, \epsilon\rangle$
$\langle s_1, r_2\rangle$
$\langle s_2, r_1\rangle$
$\langle s_3, r_1\rangle$
$\langle t_1, \epsilon\rangle$
$\langle t_2, \epsilon\rangle$
$\langle \epsilon, u_1\rangle$
$\langle \epsilon, u_1\rangle$
$\langle s_2, r_2\rangle$
$\langle s_3, r_2\rangle$
$\langle t_3, u_2\rangle$
$\langle t_4, u_2\rangle$
$\langle s_4, r_3\rangle$
$\langle t_5, \epsilon\rangle$
$\langle \epsilon, u_3\rangle$
$\langle s_1, r_3\rangle$
$\langle s_4, r_1\rangle$
$\langle t_1, \epsilon\rangle$
$\langle t_2, \epsilon\rangle$
$\langle \epsilon, u_1\rangle$
$\langle \epsilon, u_3\rangle$
$\langle t_5, \epsilon\rangle$
$\langle s_2, r_3\rangle$
$\langle s_3, r_3\rangle$
$\langle s_4, r_2\rangle$
$\langle \epsilon, u_3\rangle$

Unfolding Products

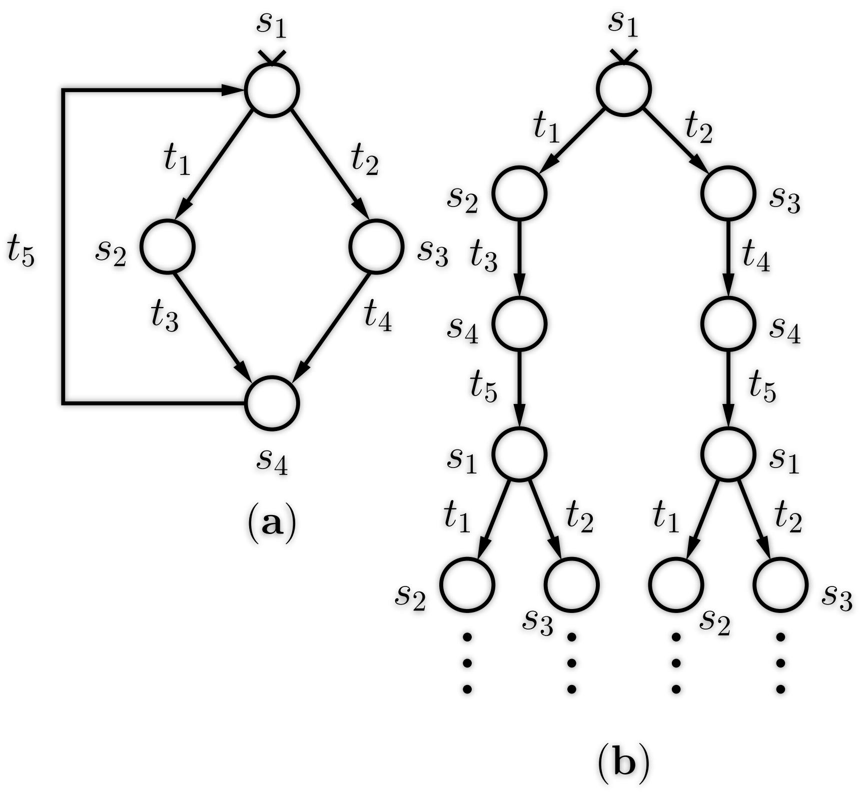
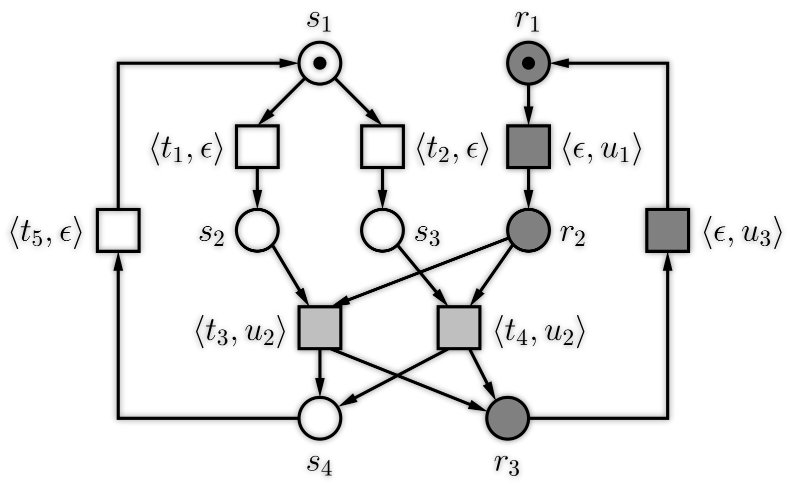
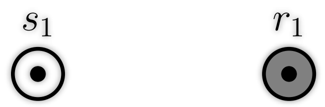


Fig. 3.1. The transition system of Fig. 2.1 (**a**) and its unfolding (**b**) as a transition system

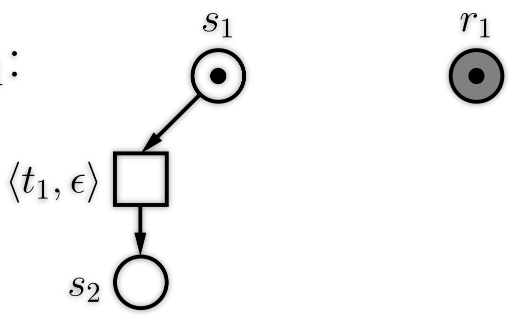
We now address the question of how to unfold a product. The answer is easy if we take the interleaving representation of products as defined in Sect. 2.4: The unfolding of a product $\mathbf{A}$ can be defined as the unfolding of the transition system $\mathcal{T}_{\mathbf{A}}$. However, in this book we investigate a different notion of unfolding, which corresponds to taking the Petri net representation of products. In Sect. 3.1 we introduce, first intuitively and then formally, the notion of *branching processes*, and the notion of *the unfolding* of a product as the "largest" branching process. In Sect. 3.2 we present some basic properties of branching processes. Section 3.3 explains why unfolding-based verification can be more efficient than verification based on the interleaving representation of products. Section 3.4 discusses the algorithmic problem of computing the unfolding. Section 3.5 introduces the notion of a search procedure for solving a verification problem. Finally, Sect. 3.6 sets the plan for the next chapters.


s_1
r_1
$\langle t_1, \epsilon \rangle$
$\langle t_2, \epsilon \rangle$
$\langle \epsilon, u_1 \rangle$
$\langle t_5, \epsilon \rangle$
s_2
s_3
r_2
$\langle \epsilon, u_3 \rangle$
$\langle t_3, u_2 \rangle$
$\langle t_4, u_2 \rangle$
s_4
r_3

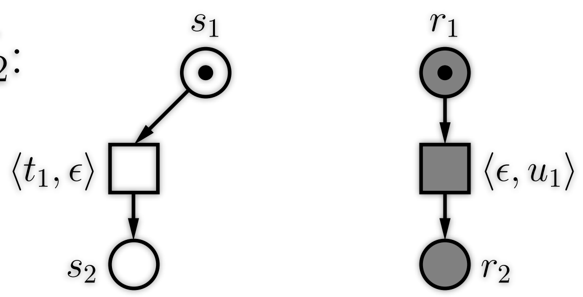
$\mathcal{N}_0$:



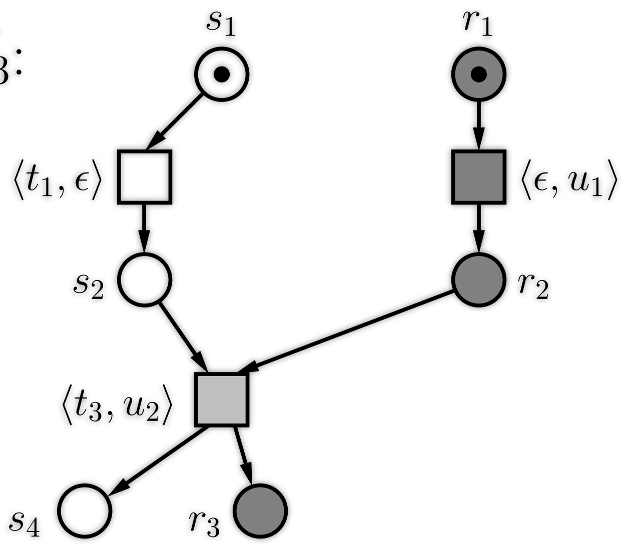
$\mathcal{N}_1$:



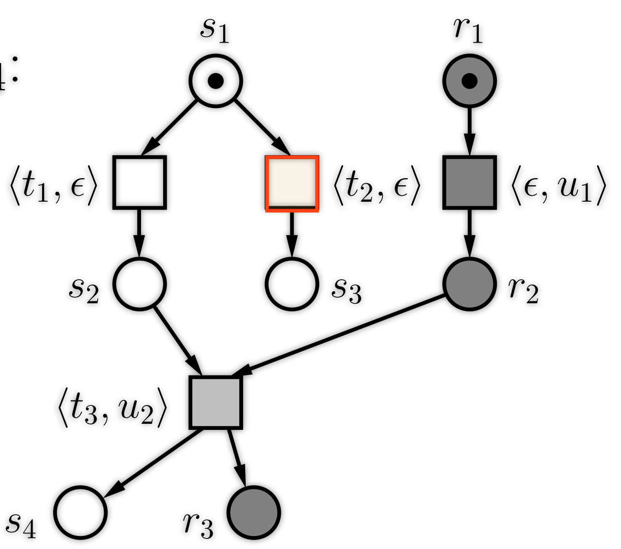
$\mathcal{N}_2$:

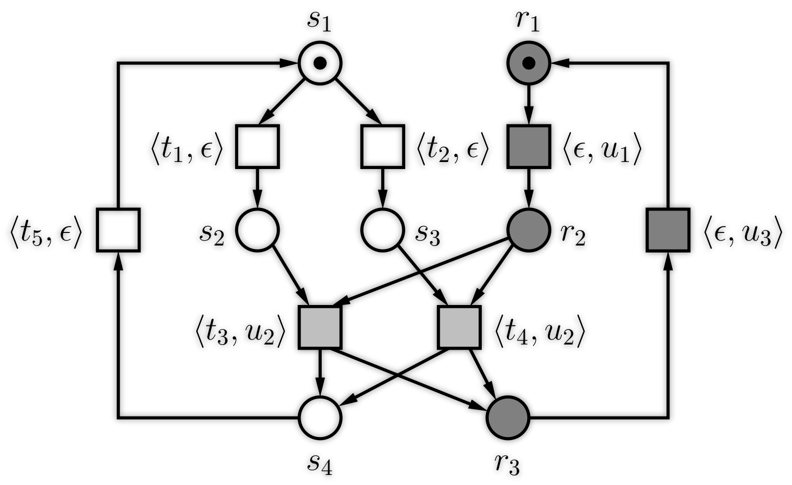

s_1
r_1
$\langle t_1, \epsilon \rangle$
$\langle \epsilon, u_1 \rangle$
s_2
r_2

$\mathcal{N}_3$:

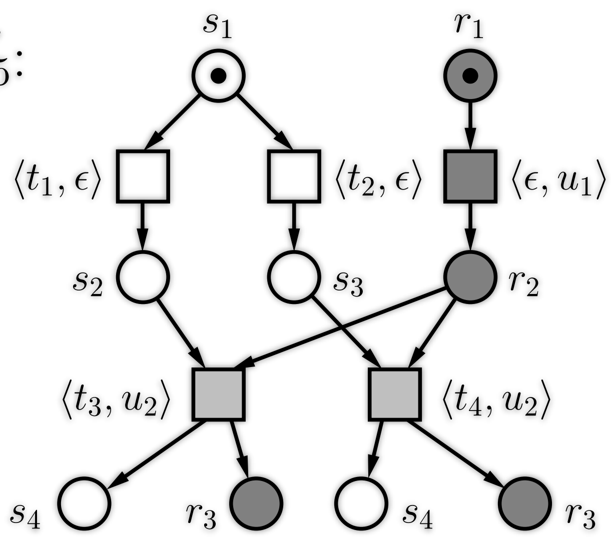

s_1
r_1
$\langle t_1, \epsilon \rangle$
$\langle \epsilon, u_1 \rangle$
s_2
r_2
$\langle t_3, u_2 \rangle$
s_4
r_3

$\mathcal{N}_4$:

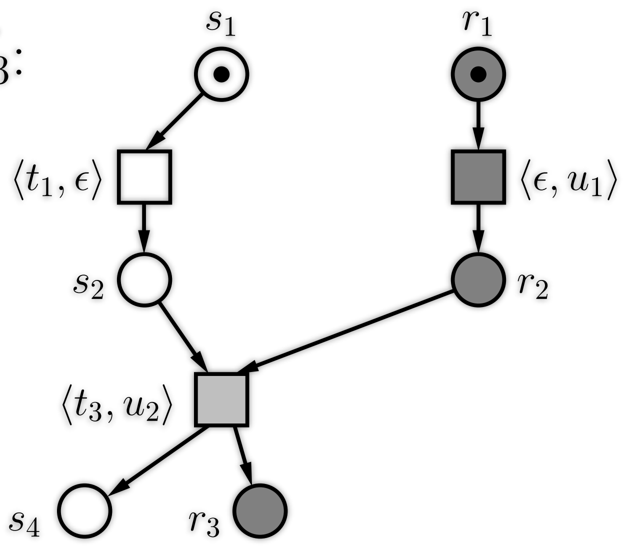

s_1
r_1
$\langle t_1, \epsilon \rangle$
$\langle t_2, \epsilon \rangle$
$\langle \epsilon, u_1 \rangle$
s_2
s_3
r_2
$\langle t_3, u_2 \rangle$
s_4
r_3


s_1
r_1
$\langle t_1, \epsilon \rangle$
$\langle t_2, \epsilon \rangle$
$\langle \epsilon, u_1 \rangle$
$\langle t_5, \epsilon \rangle$
s_2
s_3
r_2
$\langle \epsilon, u_3 \rangle$
$\langle t_3, u_2 \rangle$
$\langle t_4, u_2 \rangle$
s_4
r_3

$\mathcal{N}_5$:


s_1
r_1
$\langle t_1, \epsilon \rangle$
$\langle t_2, \epsilon \rangle$
$\langle \epsilon, u_1 \rangle$
s_2
s_3
r_2
$\langle t_3, u_2 \rangle$
$\langle t_4, u_2 \rangle$
s_4
r_3
s_4
r_3

$\mathcal{N}_3$:


s_1
r_1
$\langle t_1, \epsilon \rangle$
$\langle \epsilon, u_1 \rangle$
s_2
r_2
$\langle t_3, u_2 \rangle$
s_4
r_3

Convention 1. *In order to avoid confusion, it is convenient to use different names for the transitions of a transition system or product of transition systems, and for the transitions of its unfolding. We call the transitions of an unfolding* events. *An event corresponds to a particular* occurrence *of a transition. In the figures we use the natural numbers* $1, 2, 3, \ldots$ *as event names.*

Definition 3.2. *The* union $\bigcup N$ *of a (finite or infinite) set* N *of Petri nets is defined as the Petri net*

$$\bigcup N = \left(\bigcup_{(P,E,F,M_0)\in N} P, \bigcup_{(P,E,F,M_0)\in N} E, \bigcup_{(P,E,F,M_0)\in N} F, \bigcup_{(P,E,F,M_0)\in N} M_0 \right).$$

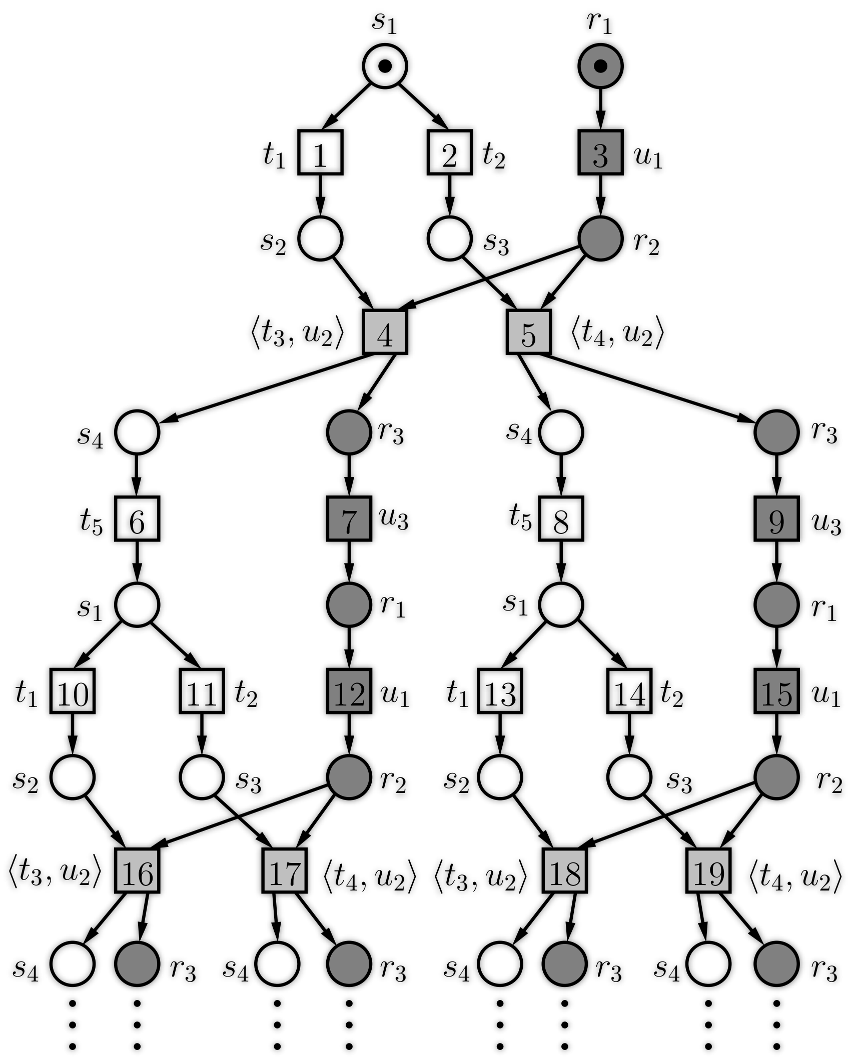
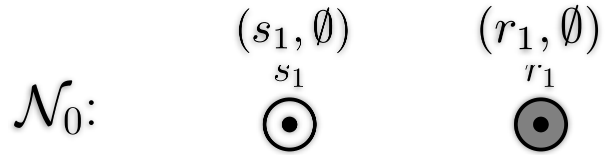
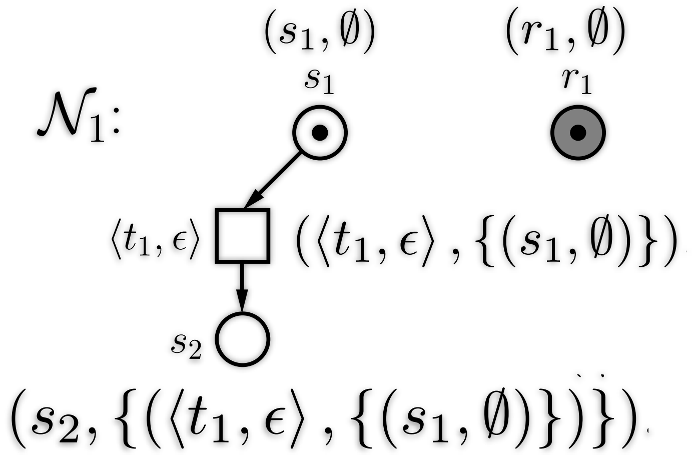


Fig. 3.3. The unfolding of the product represented in Fig. 2.4 on p. 10

We solve this problem by introducing a canonical way of naming nodes. Loosely speaking, an event labeled by a global transition $\mathbf{t} \in \mathbf{T}$ is given the name $(\mathbf{t}, X)$, where X is the set containing the names of the input places of the event. Similarly, a place labeled by a local state $s \in S_i$ is given the name $(s, \{x\})$, where x is the name of the unique input event of the place. We say that $(\mathbf{t}, X)$ and $(s, \{x\})$ are the *canonical names* of the nodes.

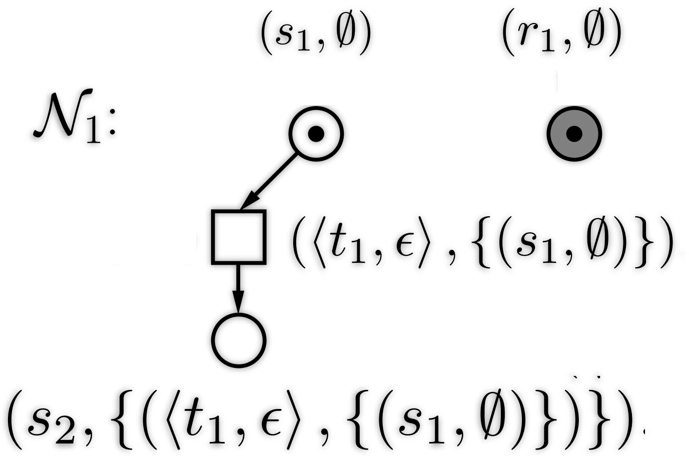



$\mathcal{N}_1$:
$(s_1, \emptyset)$
s_1
$(r_1, \emptyset)$
r_1
$\langle t_1, \epsilon \rangle$
$(\langle t_1, \epsilon \rangle, \{(s_1, \emptyset)\})$
s_2
$(s_2, \{(\langle t_1, \epsilon \rangle, \{(s_1, \emptyset)\})\})$

Formally, given a product $\mathbf{A} = \langle \mathcal{A}_1, \ldots, \mathcal{A}_n, \mathbf{T} \rangle$ we define the set $\mathcal{C}$ of *canonical names* as the smallest set satisfying the following property: if $x \in S_1 \cup \ldots \cup S_n \cup \mathbf{T}$ and X is a finite subset of $\mathcal{C}$, then $(x, X) \in \mathcal{C}$. We call x the *label* of (x, X), and say that (x, X) is *labeled* by x. Notice that $\mathcal{C}$ is nonempty, because $(x, \emptyset)$ belongs to $\mathcal{C}$ for every $x \in S_1 \cup \ldots \cup S_n \cup \mathbf{T}$.

Definition 3.4. *A $\mathcal{C}$-Petri net is a Petri net (P, E, F, M_0) such that:*

(1) $P \cup E \subseteq \mathcal{C}$,
(2) if $(x, X) \in P \cup E$, then $X = {}^{\bullet}(x, X)$; and
(3) for every $(x, X) \in P$, $(x, X) \in M_0$ if and only if $X = \emptyset$.

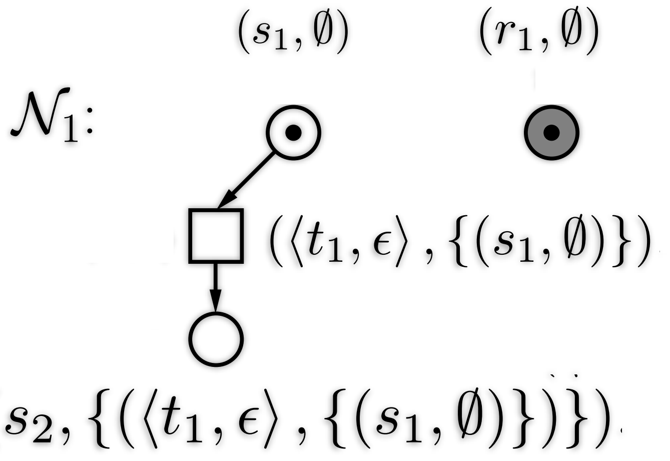

$\mathcal{N}_1$:
$(s_1, \emptyset)$
$(r_1, \emptyset)$
$(\langle t_1, \epsilon \rangle, \{(s_1, \emptyset)\})$
$(s_2, \{(\langle t_1, \epsilon \rangle, \{(s_1, \emptyset)\})\})$

Definition 3.5. *The set of* branching processes *of a product* $\mathbf{A}$ *is the smallest set of* $\mathcal{C}$*-Petri nets satisfying the following conditions:*

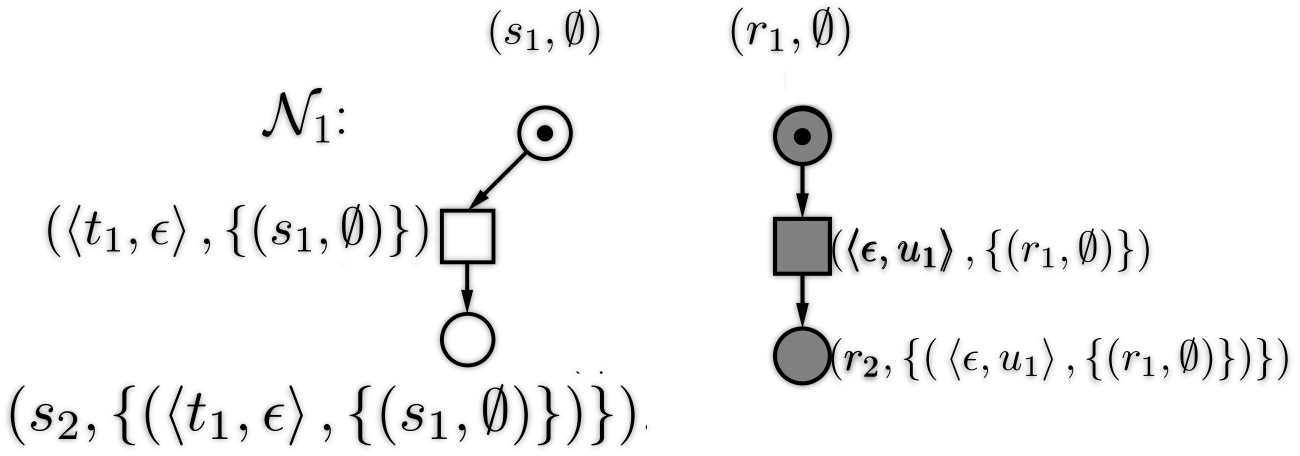
(1) Let $Is = \{(is_1, \emptyset), \ldots, (is_n, \emptyset)\}$*, where* $\{is_1, \ldots, is_n\}$ *is the set of initial states of the components of* $\mathbf{A}$*. The* $\mathcal{C}$*-Petri net having* Is *as set of places and no events is a branching process of* $\mathbf{A}$*.*

(2) Let $\mathcal{N}$ *be a branching process of* $\mathbf{A}$ *such that some reachable marking of* $\mathcal{N}$ *enables a global transition* $\mathbf{t}$*. Let* M *be the set containing the places of the marking that are labeled by* ${}^\bullet\mathbf{t}$*. The* $\mathcal{C}$*-Petri net obtained by adding to* $\mathcal{N}$ *the event* $(\mathbf{t}, M)$ *and one place* $(s, \{(\mathbf{t}, M)\})$ *for every* $s \in \mathbf{t}^\bullet$ *is also a branching process of* $\mathcal{N}$*. We call the event* $(\mathbf{t}, M)$ *a* possible extension *of* $\mathcal{N}$*.*

(3) If B *is a (finite or infinite) set of branching processes of* $\mathbf{A}$*, then so is* $\bigcup B$*.*



$\mathcal{N}_1$:
$(s_1, \emptyset)$
$(r_1, \emptyset)$
$(\langle t_1, \epsilon\rangle, \{(s_1, \emptyset)\})$
$(s_2, \{(\langle t_1, \epsilon\rangle, \{(s_1, \emptyset)\})\})$


$(s_1, \emptyset)$
$(r_1, \emptyset)$
$\mathcal{N}_1$:
$(\langle t_1, \epsilon \rangle, \{(s_1, \emptyset)\})$
$(\langle \epsilon, u_1 \rangle, \{(r_1, \emptyset)\})$
$(r_2, \{(\langle \epsilon, u_1 \rangle, \{(r_1, \emptyset)\})\})$
$(s_2, \{(\langle t_1, \epsilon \rangle, \{(s_1, \emptyset)\})\})$

In particular, Prop. 3.7 implies the existence of a very tight relation between the histories of a product **A** and the occurrence sequences of its unfolding. In order to formulate this result, we extend the labeling function λ to sequences of events. Given a finite or infinite sequence $\sigma = e_0\, e_1\, e_2 \ldots$, we define $\lambda(\sigma) = \lambda(e_0)\, \lambda(e_1)\, \lambda(e_2) \ldots$.

Corollary 3.9. *(a) If σ is a (finite or infinite) occurrence sequence of the unfolding, then $\lambda(\sigma)$ is a history of* **A**.
(b) If **h** *is a history of* **A**, *then some occurrence sequence of the unfolding satisfies $\lambda(\sigma) =$* **h**.

3.2 Some Properties of Branching Processes

We list some properties of branching processes. They can all be easily proved by structural induction on the definition of branching processes, and in most cases we only sketch their proofs.

Definition 3.10. *A place of an unfolding is an* i-place *if it is labeled by a state of the* i*th component. The* i-root *is the unique* i*-place having no input events. An event is an* i-event *if it is labeled by a global transition* $\langle t_1, \ldots, t_n \rangle$ *such that* $t_i \neq \epsilon$*. In other words, an event is an* i*-event if the* i*th component participates in the global transition it is labeled with.*

Proposition 3.11. *Let $\mathcal{N}$ be a branching process of* $\mathbf{A}$. *Then:*

(1) $\mathcal{N}$ *has no cycles, i.e., no (nonempty) path of arcs leads from a node to itself.*

(2) For every $i \in \{1, \dots n\}$, every reachable marking of $\mathcal{N}$ puts a token in exactly one i-place.

(3) The set of i-nodes of the branching process $\mathcal{N}$ forms a tree with the i-root as root. Moreover, the tree only branches at places, i.e., if a node of the tree has more than one child, then it is a place.

(4) A place of $\mathcal{N}$ can get marked at most once (i.e., if along an occurrence sequence it becomes marked and then unmarked, then it never becomes marked again), and an event of $\mathcal{N}$ can occur at most once in an occurrence sequence.

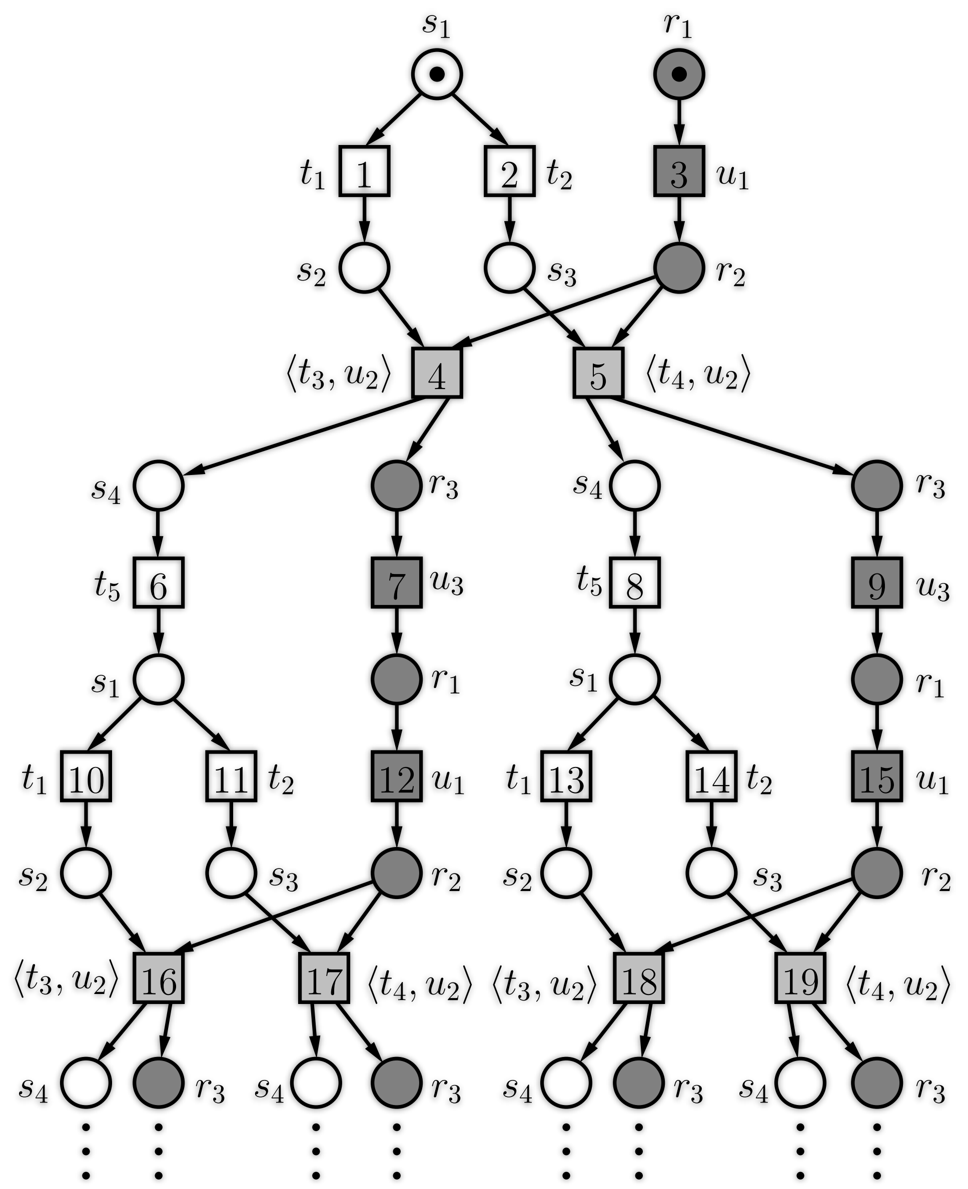


Fig. 3.3. The unfolding of the product represented in Fig. 2.4 on p. 10

3.2.2 Causality, Conflict, and Concurrency

Definition 3.13. *Let x and y be two nodes of an unfolding.*

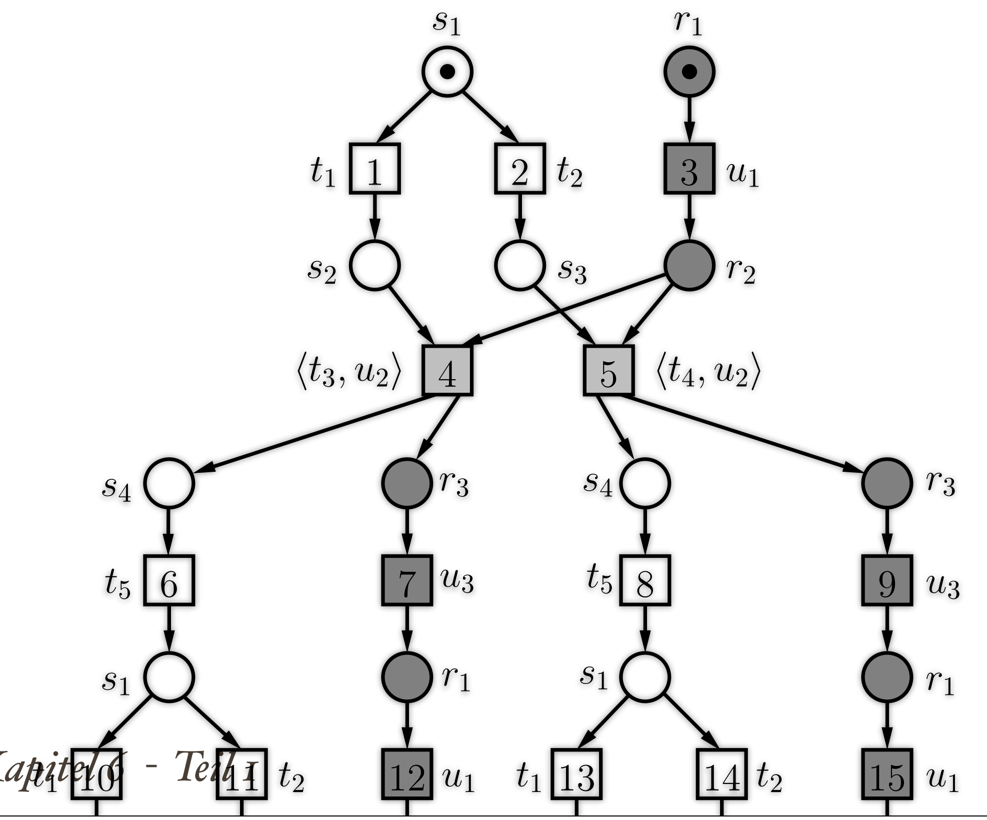
- *We say that x is a* causal predecessor *of y, denoted by $x < y$, if there is a (non-empty) path of arcs from x to y; as usual we denote by $x \leq y$ that either $x < y$ or $x = y$; two nodes x and y are* causally related *if $x \leq y$ or $x \geq y$.*
- *We say that x and y are in* conflict*, denoted by $x\#y$, if there is a place z, different from x and y, from which one can reach x and y, exiting z by different arcs.*
- *We say that x and y are* concurrent*, denoted by x co y, if x and y are neither causally related nor in conflict.*

Proposition 3.14. *Let $\mathcal{N}$ be a branching process of $\mathbf{A}$ and let P be a set of places of $\mathcal{N}$. There is a reachable marking M of $\mathcal{N}$ such that $P \subseteq M$ if and only if the places of P are pairwise concurrent.*

Proposition 3.15. *(1) For every two nodes x, y of a branching process exactly one of the following holds: (a) x and y are causally related, (b) x and y are in conflict, (c) x and y are concurrent.*
(2) If x and y are causally related and $x \neq y$, then either $x < y$ or $y < x$, but not both.

3.2.3 Configurations

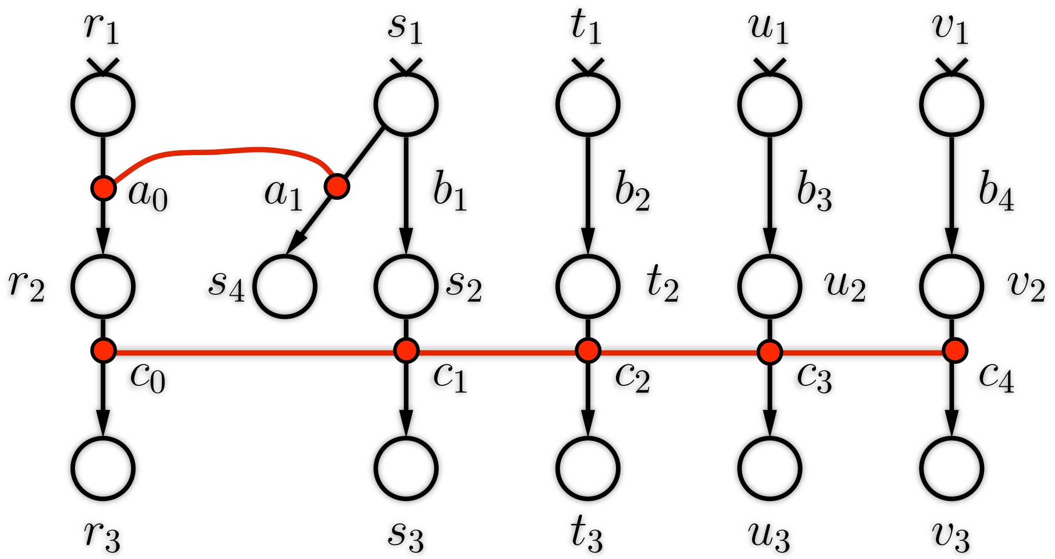
A *realization* of a set of events is an occurrence sequence of the branching process in which each event of the set occurs exactly once, and no other events occur. A set of events can have zero, one, or more realizations. For instance, the sets $\{1, 2\}$ and $\{4, 6\}$ in Fig. 3.3 on p. 17 have no realizations (for the latter, recall that occurrence sequences start at the initial marking, which enables neither event 4 nor event 6), and the set $\{1, 3, 4, 7\}$ has two realizations, namely the sequences 1 3 4 7 and 3 1 4 7.



Definition 3.17. *A set of events of an unfolding is a* configuration *if it has at least one realization.*

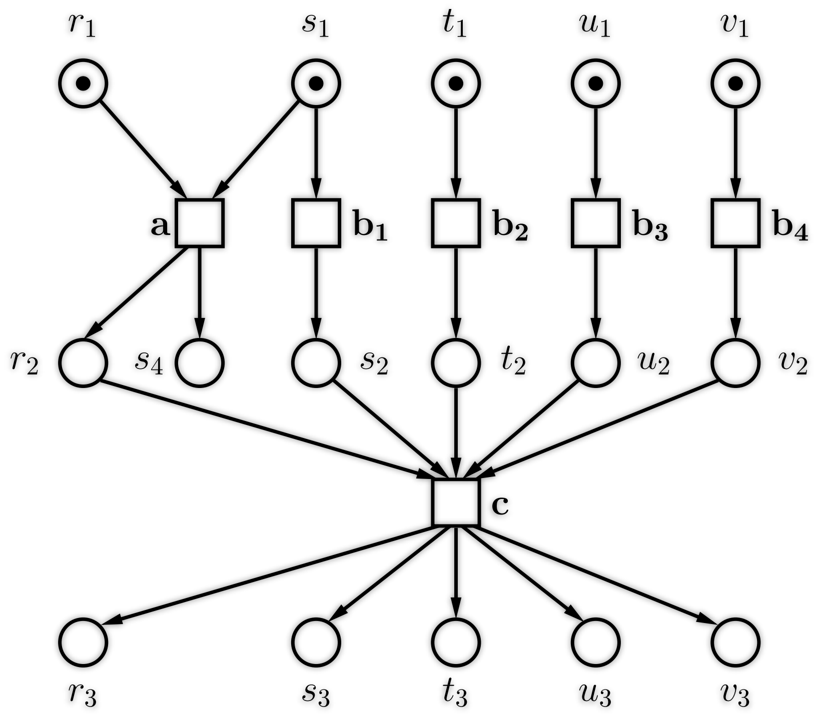
Proposition 3.18. *Let $\mathcal{N}$ be a branching process of a product* $\mathbf{A}$ *and let E be a set of events of $\mathcal{N}$.*

(1) E is a configuration if and only if it is causally closed*, i.e., if $e \in E$ and $e' < e$ then $e' \in E$, and* conflict-free*, i.e., no two events of E are in conflict.*
(2) All the realizations of a finite configuration lead to the same reachable marking of $\mathcal{N}$.



$$\begin{aligned}\mathbf{T} = \{\mathbf{a} = \langle a_0, a_1, \epsilon, \epsilon, \epsilon\rangle, \mathbf{b_1} = \langle \epsilon, b_1, \epsilon, \epsilon, \epsilon\rangle,\\ \mathbf{b_2} = \langle \epsilon, \epsilon, b_2, \epsilon, \epsilon\rangle, \mathbf{b_3} = \langle \epsilon, \epsilon, \epsilon, b_3, \epsilon\rangle,\\ \mathbf{b_4} = \langle \epsilon, \epsilon, \epsilon, \epsilon, b_4\rangle, \mathbf{c} = \langle c_0, c_1, c_2, c_3, c_4\rangle\}\end{aligned}$$

We wish to know if the global transition $\mathbf{c} = \langle c_0, \ldots, c_4\rangle$ is executable.



The transition system has 24 global states and 40 transitions,

while the unfolding has 11 places and five events.

bei n *Transitionen:*

$2n + 1$ places and n events

transition system has $3 \cdot 2^{n-2}$ global states

Proposition 3.21. *Let $\mathcal{N}$ be a branching process of a product $\mathbf{A}$, and let $\mathbf{t}$ be a global transition of $\mathbf{A}$. Deciding whether $\mathcal{N}$ can be extended with an event labeled by $\mathbf{t}$ is NP-complete.*

Proof. For membership in NP we guess a global transition $\mathbf{t}$ and a set of places of the unfolding $M = \{p_1, \ldots, p_k\}$ labeled by ${}^\bullet\mathbf{t} = \{s_1, \ldots, s_k\}$, and check in polynomial time, using the procedure sketched above, that its input places are pairwise concurrent. After this we still need to check in polynomial time that no event exists in the unfolding labeled with $\mathbf{t}$ and having preset M to ensure the event is a proper extension of the unfolding.

We prove NP-hardness by a reduction from CNF-3SAT formula over variables $x_1, x_2, \ldots, x_n$. A literal is either a variable x_i or its negation $\overline{x}_i$. Let $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ be a CNF-3SAT formula, where each conjunct C_j is a disjunction of at most three literals. We construct in polynomial time a product $\mathbf{F}$ of transition systems defined as follows:

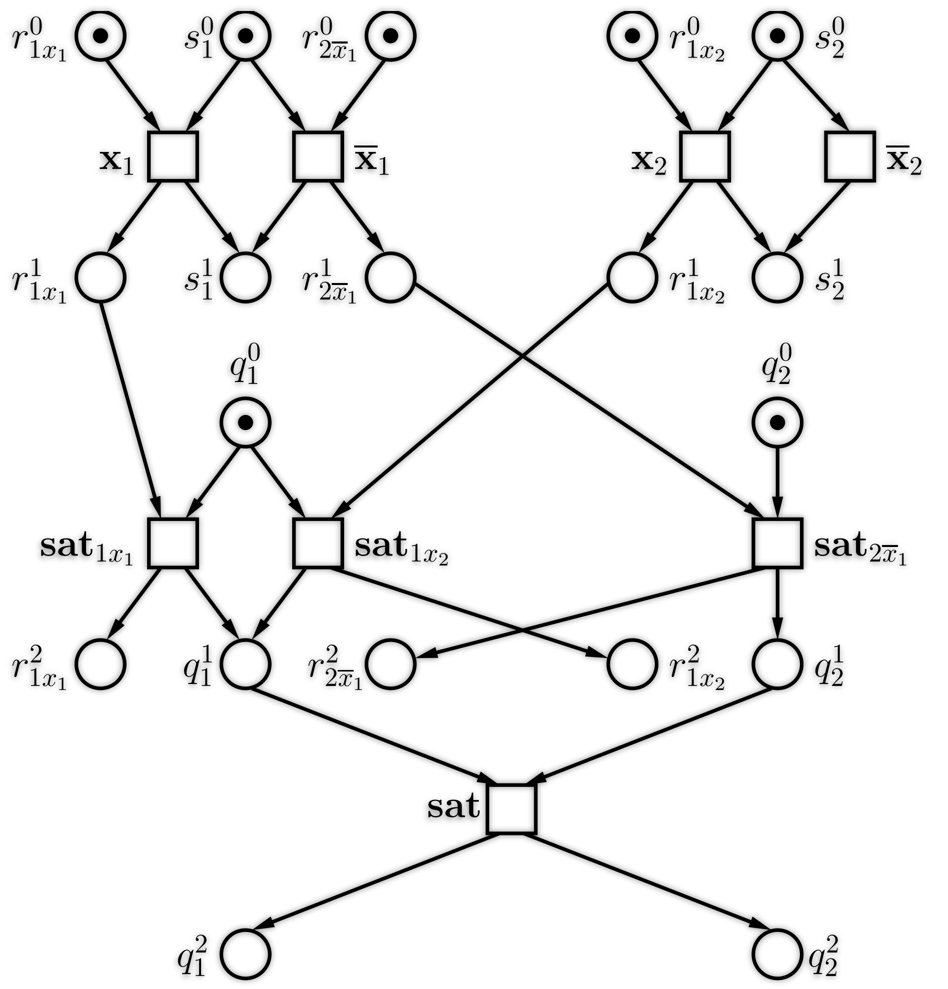


Fig. 3.9. Petri net representation of the product $\mathbf{F}$ for $F = (x_1 \vee x_2) \wedge \overline{x}_1$

Consider now the prefix of the unfolding of **F** obtained by removing from the full unfolding all events labeled by **sat**. This prefix can be easily constructed in polynomial time in the size of **F** because all global transitions (except for **sat**) have a bounded synchronization degree. The prefix can be extended with an event labeled by **sat** if and only if the formula F is satisfiable. $\square$

3.5 Search Procedures

In this book we consider verification questions of the form: "Does the system have a (possibly infinite) history satisfying a given property?" Our computational approach consists of computing larger and larger prefixes of the unfolding, until we have enough information to answer the question. The prefixes are generated by *search procedures*.

A search procedure consists of a *search scheme* and a *search strategy*. The search strategy determines, given the current prefix of the unfolding, which event should be added to it next. Notice that a strategy may be nondeterministic, i.e., it may decide that any element out of the set of possible extensions should be added next. Depth-first and breadth-first are typical strategies for transition systems. The search scheme depends on the property we are interested in. It determines which leaves of the prefix need not be explored further, and whether the search is successful. More precisely, a search scheme consists of two parts:

- A *termination condition* determining which leaves of the current prefix are *terminals*, i.e., nodes whose causal successors need not to be explored.[5]
- A *success condition* determining which terminals are *successful*, i.e., terminals proving that the property holds.

```
procedure unfold(product A) {
   $\mathcal{N}$ := unique branching process of A without events;
   $T := \emptyset$; $S := \emptyset$; $X := Ext(\mathcal{N}, T)$;
   while ($X \neq \emptyset$) {
      choose an event $e \in X$ according to the search strategy;
      extend $\mathcal{N}$ with $e$;
      if $e$ is a terminal according to the search scheme then {
         $T := T \cup \{e\}$;
         if $e$ is successful according to the search scheme then {
            $S := S \cup \{e\}$; /* A successful terminal found */
         };
      };
      $X := Ext(\mathcal{N}, T)$;
   };
   return $\langle \mathcal{N}, T, S \rangle$;
};
```

T the set of terminal events

S set of successful

Fig. 3.10. Pseudo-code of the unfolding procedure

$Ext(\mathcal{N}, T)$ denotes the set of events that can be added to $\mathcal{N}$ according to Def. 3.5 on p. 18 and have no causal predecessor in the set of terminal events T.

Given a product **A** and a terminating search procedure P, the *final prefix* is the prefix generated by P on input **A** after termination. The final prefix is *successful* if it contains at least one successful terminal. Given a property ϕ, a terminating procedure P is *complete* if the final prefix it generates is successful for every product **A** satisfying ϕ, and *sound* if every product **A** such that the final prefix is successful satisfies ϕ.

The ultimate goal of this book is to present a search procedure for model checking a product **A** against arbitrary properties expressed in Linear Temporal Logic (LTL), a popular specification language.^{6} The search procedure is presented in Chap. 8. It is based on search procedures for three central verification problems which are also interesting on their own:

- **The executability problem**: Given a set $\mathbf{G} \subseteq \mathbf{T}$ of global transitions, can some transition of $\mathbf{G}$ ever be executed, i.e., is there a global history whose last event is labeled by an element of $\mathbf{G}$?

- **The repeated executability problem**: Given a set $\mathbf{R} \subseteq \mathbf{T}$ of global transitions, can some transition of $\mathbf{R}$ be executed infinitely often, i.e., is there an infinite global history containing infinitely many events labeled by transitions of $\mathbf{R}$?

- **The livelock problem**: Given a partitioning of the global transitions into *visible* and *invisible*, and given a set $\mathbf{L} \subseteq \mathbf{T}$ of visible transitions, is there an infinite global history in which a transition of $\mathbf{L}$ occurs, followed by an infinite sequence of invisible transitions?

Theorem 3.24. *The executability, repeated executability, and livelock problems are PSPACE-complete for products.*

Proof. (Sketch.) We only consider the executability problem, since the proof for the other two problems is similar. To prove membership in PSPACE we observe first that, since NPSPACE=PSPACE by Savitch's theorem (see, e.g., [98]), it suffices to provide a nondeterministic algorithm for the problem using only polynomial space. The algorithm uses a variable $\mathbf{v}$ to store one global state; initially, $\mathbf{v} = \mathbf{is}$. While $\mathbf{v} \neq \mathbf{s}$, the algorithm repeatedly selects a global transition $\mathbf{t}$ enabled at $\mathbf{v}$, computes the global state $\mathbf{s}'$ such that $\langle \mathbf{v}, \mathbf{t}, \mathbf{s}' \rangle$, and sets $\mathbf{v} := \mathbf{s}'$. If at some point $\mathbf{v} = \mathbf{s}$, the algorithm stops and outputs the result "reachable". Obviously, the algorithm only needs linear space.

PSPACE-hardness is proved by reduction from the following problem: given a polynomially space-bounded Turing machine M and an input x, does M accept x? We assume w.l.o.g. that M has one single accepting state q_f.

Let $p(n)$ be the polynomial limiting the number of tape cells that M uses on an input of length n. We construct a product $\mathbf{A} = \langle \mathcal{A}_Q, \mathcal{A}_1, \ldots, \mathcal{A}_{p(|x|)}, \mathbf{T} \rangle$. The component $\mathcal{A}_Q$ contains one state s_q for each control state q of M. The intended meaning of s_q is that the machine M is currently in state q. For every $i \in \{1, \ldots, p(|x|)\}$ and for every tape symbol a of M, the component $\mathcal{A}_i$ contains two states $s_{\langle a,0 \rangle}$ and $s_{\langle a,1 \rangle}$. The intended meaning of $s_{\langle a,0 \rangle}$ is: the ith tape cell currently contains the symbol a, and the head of M is not reading the cell. The intended meaning of $s_{\langle a,1 \rangle}$ is: the ith tape cell currently contains the symbol a, and the head of M is reading the cell. The initial states of the component are chosen according to the initial configuration of M: the initial state of $\mathcal{A}_Q$ is s_{q_0}, where q_0 is the initial state of M; the initial state of component $\mathcal{A}_1$ is $s_{\langle x_1,1 \rangle}$, where x_1 is the first letter of the input x; and so on. The transitions of the components and the synchronization vector $\mathbf{T}$ are chosen so that the execution of a global transition of $\mathbf{A}$ corresponds to a move of M. Additionally, the component $\mathcal{A}_q$ has a transition t_f with s_{q_f} as both source and target state, and $\mathbf{T}$ contains a synchronization vector $\mathbf{t}_f = \langle t_f, \epsilon, \ldots, \epsilon \rangle$.

Clearly, M halts for input x if and only if the instance of the executability problem for $\mathbf{A}$ given by $\mathbf{G} = \{\mathbf{t}_f\}$ has a positive answer. $\square$