

Search Procedures for the Executability Problem

We fix a transition system $\mathcal{A} = \langle S, T, \alpha, \beta, is \rangle$ and a set $G \subseteq T$ of *goal transitions*. We wish to solve the problem of whether some history of \mathcal{A} executes some transition of G by means of a finite, sound, and complete search procedure. It is not difficult to see that such procedures exist; for instance depth-first or breadth-first search will do the job. However, we wish to prove a stronger result, namely the existence of a search *scheme* that leads to a terminating, sound, and complete search procedure *for every search strategy*. In this section we formalize the notion of strategy, and in the next one we proceed to define the search scheme.

To define search strategies, it is convenient to define the notion of *order*:

Definition 4.1. *An order is a relation that is both irreflexive and transitive.*

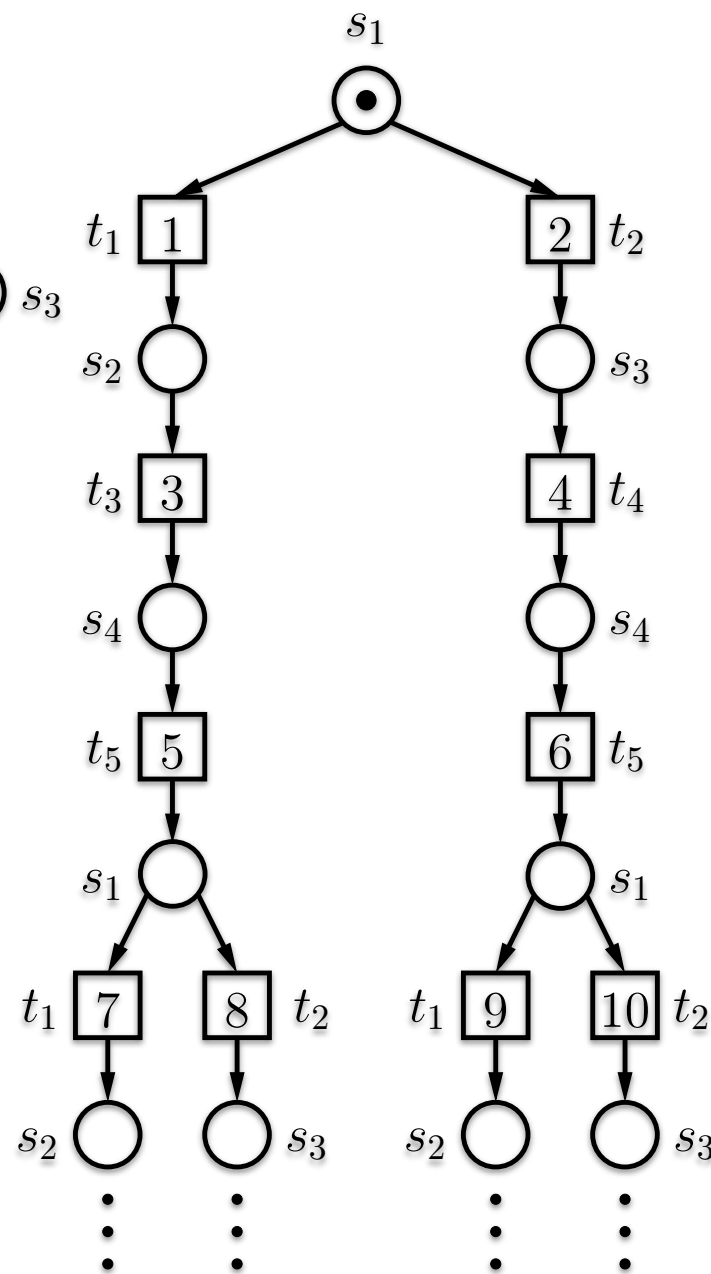
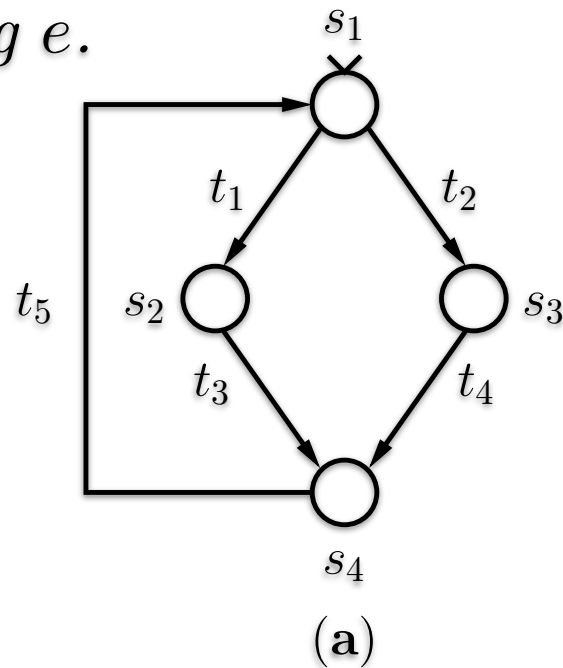
Orders are often called strict partial orders in the literature but we use the term *order* for brevity.

Recall that, loosely speaking, a search strategy determines, given the current prefix, which of its possible extensions should be added to it next. So, in full generality, a search strategy can be defined as a priority relation or as an order between branching processes. Assume the current branching process is \mathcal{N} with possible extensions e_1, \dots, e_k , and for every $i \in \{1, \dots, k\}$ let \mathcal{N}_i be the result of adding e_i to \mathcal{N} . Then the event e_j such that \mathcal{N}_j has the highest priority among $\mathcal{N}_1, \dots, \mathcal{N}_k$ is the one selected to extend \mathcal{N} . In terms of orders, we select an event e_j such that \mathcal{N}_j is minimal according to the order.

Given two events e_1, e_2 , there can be many different branching processes having e_1 and e_2 as possible extensions. For some of them the priority relation may prefer e_1 to e_2 , while for others it may be the other way round. Such strategies can be called “context-dependent”, since the choice between e_1 and e_2 does not only depend on e_1 and e_2 themselves, but also on their context. For simplicity, we restrict our attention to “context-free” strategies in which the choice between e_1 and e_2 depends only on the events themselves.

Notice that, from a computational point of view, it does not make sense to define strategies as priority relations (i.e., orders) *on events*. The reason is that the events are what the search procedure has to compute, and so the procedure does not know them in advance. To solve this problem we introduce the notion of an event’s history.

Definition 4.2. Let e be an event of the unfolding of \mathcal{A} , and let $e_1 e_2 \dots e_m$ be the unique occurrence sequence of the unfolding and ending with e , i.e., $e_m = e$. The history of e , denoted by $H(e)$, is the computation $t_1 t_2 \dots t_m$, where t_i is the label of e_i . We call the events e_1, \dots, e_{m-1} the causal predecessors of e_m . We denote by $e' < e$ that e' is a causal predecessor of e . The state reached by $H(e)$, denoted by $St(e)$, is defined as $\beta(e)$, i.e., as the state reached after executing e .



$$H(7) = t_1 t_3 t_5 t_1,$$

$$St(7) = s_2.$$

$$H(7) = H(3) t_5 t_1.$$

predecessors of event 7 are the events 1, 3, and 5.

Proposition 4.4. *An event is characterized by its history, i.e., $e = e'$ holds if and only if $H(e) = H(e')$.*

This proposition allows us to define strategies as orders on the set of all words, i.e., as orders $\prec \subseteq T^* \times T^*$. Since histories are words and events are characterized by their histories, every order on words induces an order on events. Moreover, since the set T is part of the input to the search procedure, it makes perfect computational sense to define a strategy like “choose among the possible extensions to the current prefix anyone having a shortest history”.

Abusing notation, the order on events induced by an order \prec on words is also denoted by \prec . The order need not be total (i.e., there may be distinct events e, e' such that neither $e \prec e'$ nor $e' \prec e$ holds). However, we require that \prec *refines* the prefix order on T^* , i.e., for every $w, w' \in T^*$, if w is a proper prefix of w' , then $w \prec w'$.¹ The reason is that if $H(e)$ is a proper prefix of $H(e')$ then e' can only be added to the unfolding after e , and so e' should have lower priority than e .

Definition 4.5. *A search strategy on T^* is an order on T^* that refines the prefix order.*

Notice that e is a causal predecessor of e' if and only if $H(e)$ is a proper prefix of $H(e')$. Therefore, if $e < e'$ then $H(e) \prec H(e')$ and so $e \prec e'$. We say that a search strategy *refines the causal order* on events.

4.2 Search Scheme for Transition Systems

We are ready to present a search scheme for the executability problem that is sound and complete for every search strategy. A search scheme is determined by its terminals and successful terminals, which we now define. The definition of terminals may look circular at first sight (terminals are defined in terms of the auxiliary notion of feasible events, and vice versa) but, as we shall see, it is not.

Definition 4.6. *Let \prec be a search strategy. An event e is feasible if no event $e' \prec e$ is a terminal. A feasible event e is a terminal if either*

- (a) it is labeled with a transition of G , or*
- (b) there is a feasible event $e' \prec e$, called the companion of e , such that $St(e') = St(e)$.*

A terminal is successful if it is of type (a). The \prec -final prefix is the prefix of the unfolding of A containing the feasible events.

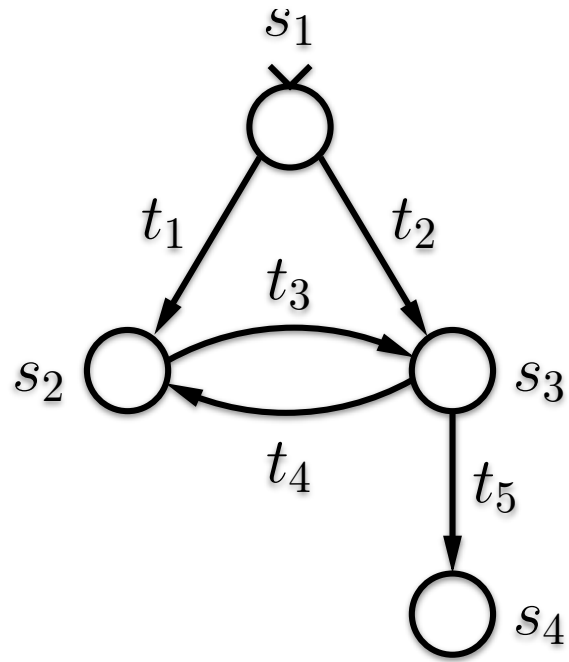
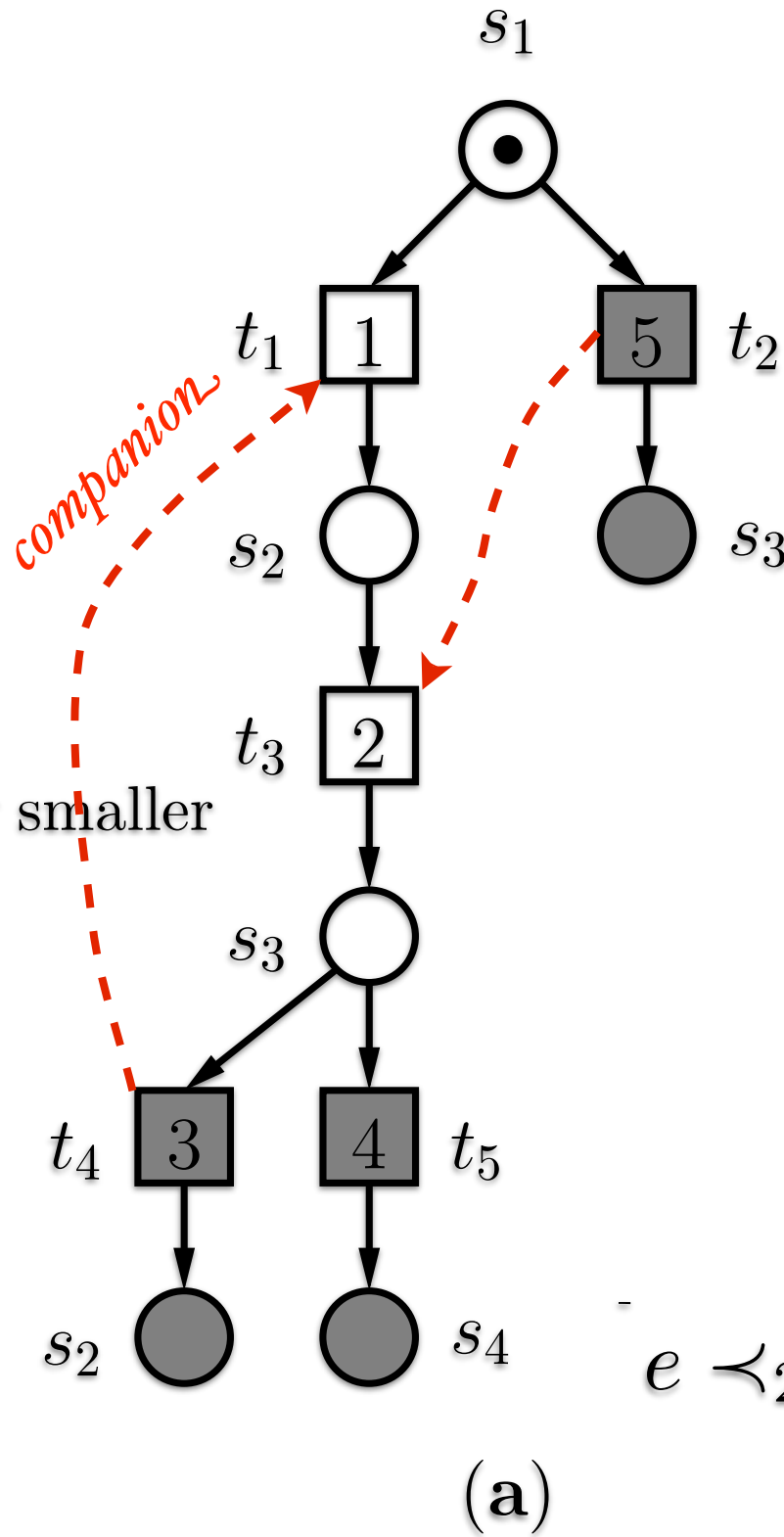
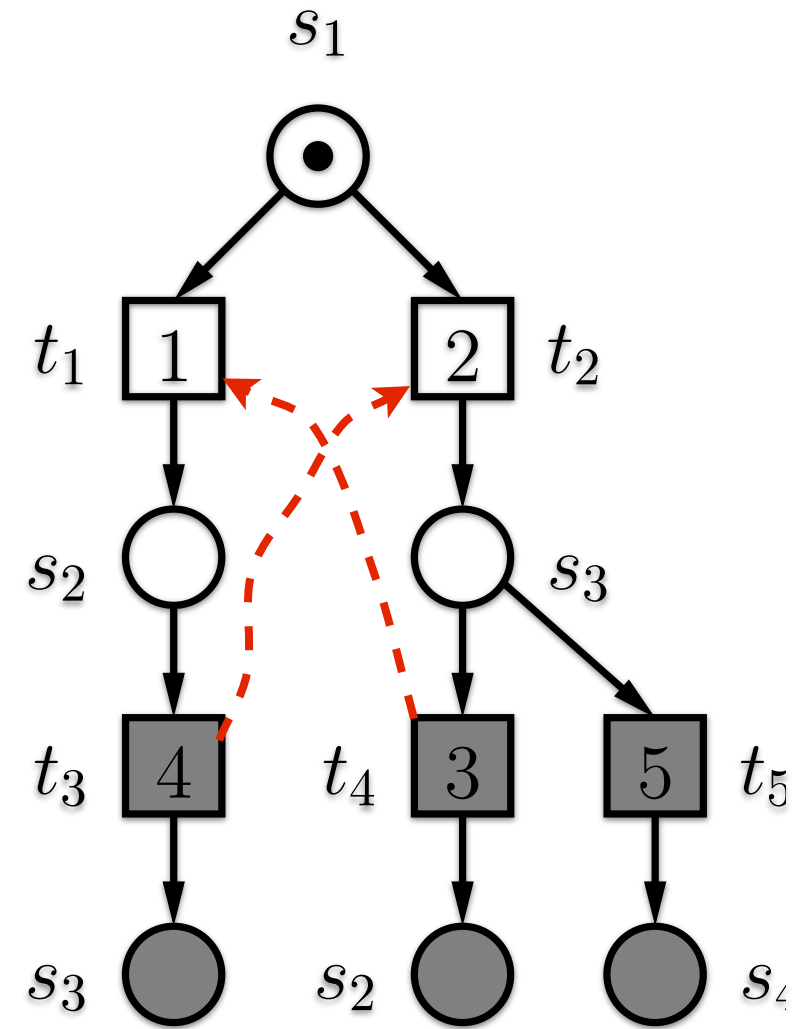


Fig. 4.1. A transition system

$e \prec_1 e'$ if $H(e)$ is lexicographically smaller



(a)



(b)

$e \prec_2 e'$ if $|H(e)| < |H(e')|$.

In order to prove that the set of terminal events is well-defined we need a lemma.

Lemma 4.8. *Let \prec be an arbitrary search strategy, and let (F, T) be a pair of sets of events satisfying the conditions of Def. 4.6 for the sets of feasible and terminal events, respectively. Then for every feasible event $e \in F$ the history $H(e)$ has length at most $|S| + 1$.*

Proof. Assume that e is a feasible event such that the length of $H(e)$ is larger than $|S| + 1$. Then, by the pigeonhole principle, there are two events $e_1 < e_2 < e$ such that $St(e_1) = St(e_2)$. Since $e \in F$, no event $e' < e$ belongs to T , and so the same holds for e_1 and e_2 . It follows $e_1, e_2 \in F$. Since $e_1 < e_2$ and the search strategy \prec refines the causal order, we have $e_1 \prec e_2$, and, by condition (b), $e_2 \in T$. Since $e_2 < e$, the event e cannot be feasible, contradicting the assumption. \square

A direct corollary of the proof above is that for all non-terminal events e the length of $H(e)$ is at most $|S|$.

Proposition 4.9. *The search scheme of Def. 4.6 is well-defined for every strategy \prec , i.e., there is a unique set of feasible events and a unique set of terminal events satisfying the conditions of the definition. Moreover, the \prec -final prefix is finite.*

Proposition 4.10. *The search scheme of Def. 4.6 is sound for every strategy.*

We now show that the search scheme is also complete *for every search strategy*. We present the proof in detail, because all the completeness proofs in the rest of the book reuse the same argumentation. It proceeds by contradiction: it is assumed that the product satisfies the property, but the final prefix is not successful. First, a set of *witnesses* is defined; these are the events of the unfolding “witnessing” that the property holds, i.e., if the search algorithm would have explored any of them then the search would have been successful. Second, an order on witnesses is defined; it is shown that the order has at least one minimal element e_m , and, using the assumption that the search was not successful, a new event e'_m is constructed. Third, it is shown that e'_m must be smaller than e_m w.r.t. the order on witnesses, contradicting the minimality of e_m .

Theorem 4.11. *The search scheme of Def. 4.6 is complete for every strategy.*

Proof. Let \prec be an arbitrary search strategy. Assume that some goal transition $g \in G$ is executable, but no terminal of the \prec -final prefix is successful. We derive a contradiction in three steps.

Witnesses. Let an event of the unfolding of \mathcal{A} be a *witness* if it is labeled with g . Since g is executable, the unfolding of \mathcal{A} contains witnesses. However, no witness is feasible, because otherwise it would be a successful terminal. So for every witness e there is an unsuccessful terminal $e_s \prec e$. We call e_s the *spoiler* of e . (see Fig. 4.3).

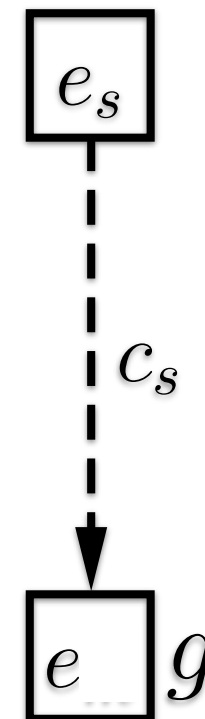
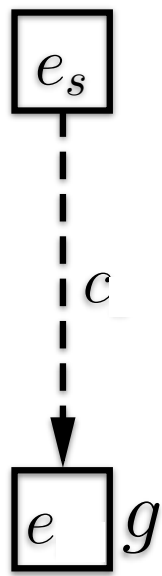


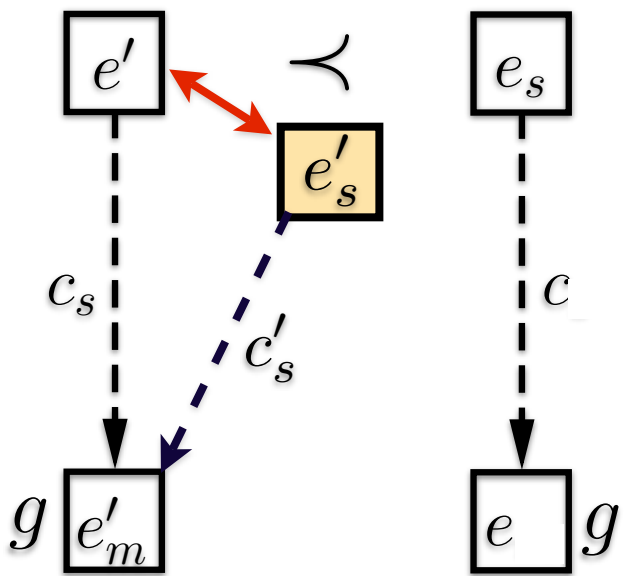
Fig. 4.3. Illustration of the proof of Thm. 4.11



Minimal witnesses. Let e be a witness and let e_s be its spoiler. Since $e_s < e$, some computation c satisfies $H(e_s)c = H(e)$. Let $l(e)$ denote the (finite) length of c . We define an order \ll on witnesses as follows: $e \ll e'$ if either $l(e) < l(e')$ or $l(e) = l(e')$ and $e_s \prec e'_s$.

We claim that \ll is well-founded. Assume this is not the case. Then there is an infinite decreasing chain of witnesses $e_w^1 \gg e_w^2 \gg e_w^3 \dots$, and because $l(e_w^i)$ can only decrease a finite number of times, we must from some index j onwards have an infinite decreasing chain of spoilers $e_s^j \succ e_s^{j+1} \succ e_s^{j+2} \dots$. Thus, since spoilers are terminals, the set of terminals must be infinite. So the \prec -final prefix is infinite, contradicting Prop. 4.9. This proves the claim.

Since \ll is well-founded, there is at least one \ll -minimal witness e_m . Let e_s be the spoiler of e_m and let c_s be the unique computation satisfying $H(e_m) = H(e_s)c_s$. Notice that, since e_m is labeled by g , the computation c_s ends with g . Since e_s is an unsuccessful terminal, it has a companion $e' \prec e_s$ such that $St(e') = St(e_s)$. Since $St(e') = St(e_s)$, both $H(e_s)c_s$ and $H(e')c_s$ are histories of \mathcal{A} . Let e'_m be the event having $H(e')c_s$ as history, i.e., $H(e'_m) = H(e')c_s$. Since c_s ends with g , the event e'_m is also labeled by g , like e_m . So e'_m is a witness, and has a spoiler e'_s . Let c'_s be the computation satisfying $H(e'_m) = H(e'_s)c'_s$.

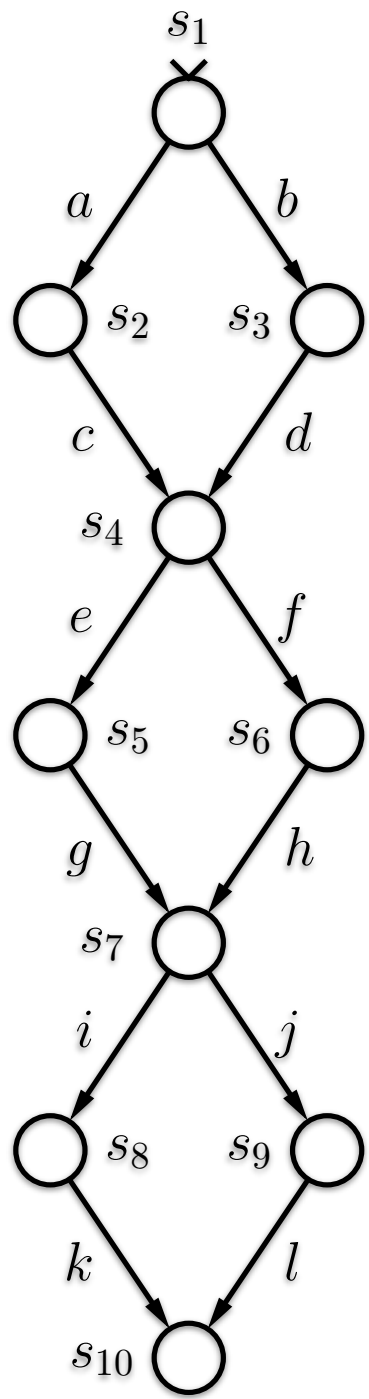


Contradiction. Since $H(e'_s)c'_s = H(e'_m) = H(e')c_s$, we have $e' < e'_m$ and $e'_s < e'_m$. So there are **three possible cases:**

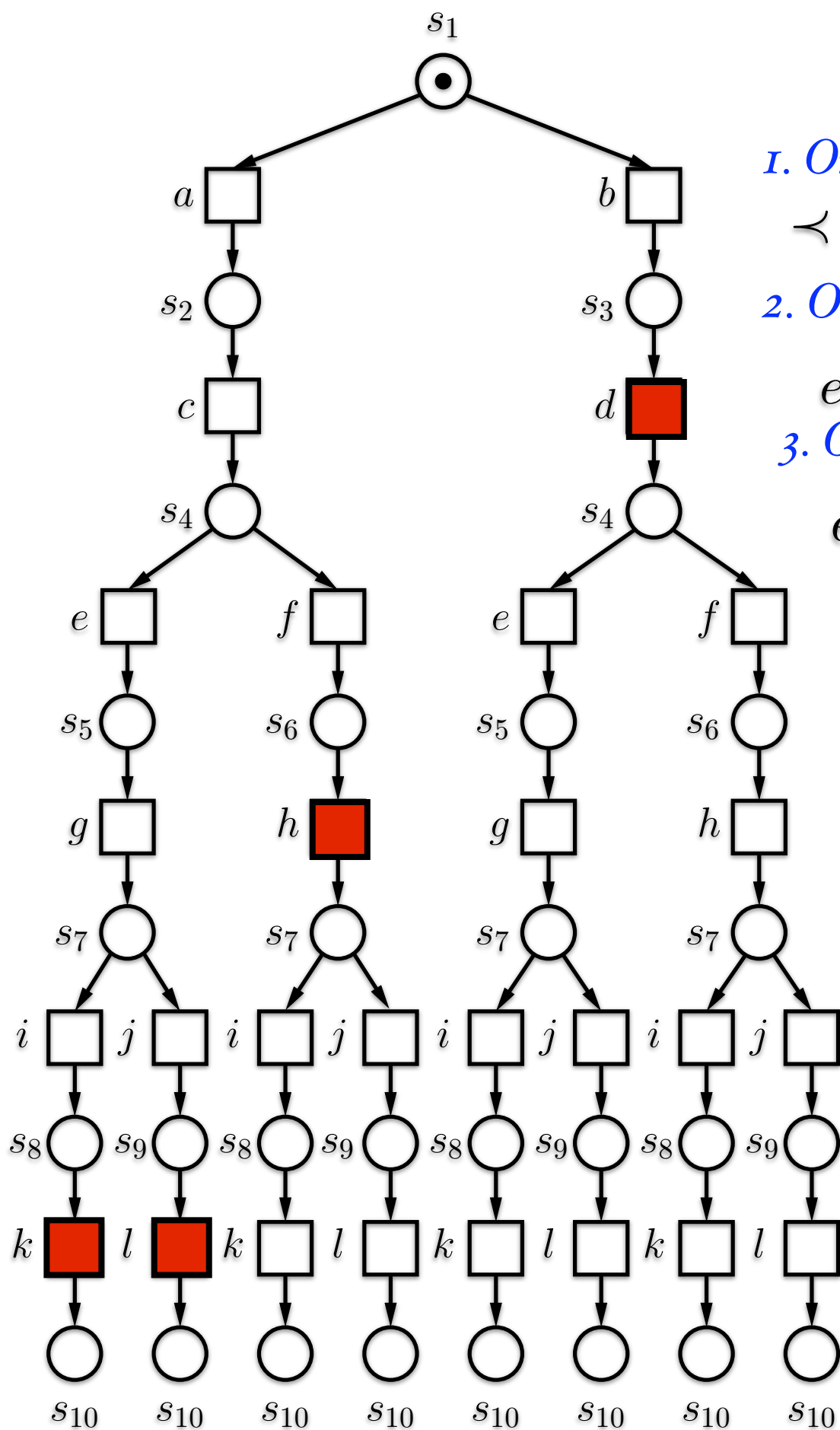
- **$e'_s < e'$.** Then, since e'_s is a spoiler and spoilers are terminals, e' is not feasible, contradicting our assumption that e' is the companion of e_s , which according to Def. 4.6 requires e' to be feasible.
- **$e'_s = e'$.** Then, since $H(e'_s)c'_s = H(e'_m) = H(e')c_s$, we have $c_s = c'_s$. Moreover, since $e'_s = e'$ and $e' \prec e_s$ we have $e'_s \prec e_s$. This implies $e'_m \ll e_m$, contradicting the minimality of e_m .
- **$e' < e'_s$.** Then, since $H(e'_s)c'_s = H(e'_m) = H(e')c_s$, the computation c'_s is shorter than c_s , and so $e'_m \ll e_m$, contradicting the minimality of e_m .

□

The next example shows that the size of the final prefix depends on the choice of strategy. In the worst case, the final prefix can be exponentially larger than the transition system.



(a)



(b)

$$G = \emptyset.$$

1. Ordnung

\prec as the prefix order on histories,

2. Ordnung

$$e \prec e' \Leftrightarrow |H(e)| < |H(e')|;$$

3. Ordnung

$e \prec e'$ if and only if $H(e)$

is lexicographically smaller than $H(e')$

One way to make the final prefix smaller is to require the strategy \prec to be a total order.

Theorem 4.13. *If \prec is a total order on T^* , then the \prec -final prefix of Def. 4.6 has at most $|S|$ feasible non-terminal events.*

Proof. If \prec is total, then, by condition (b) in the definition of a terminal, we have $St(e) \neq St(e')$ for any two feasible non-terminal events e and e' . So the final prefix contains at most as many non-terminal events as there are states in \mathcal{A} . □

4.3 Search Strategies for Products

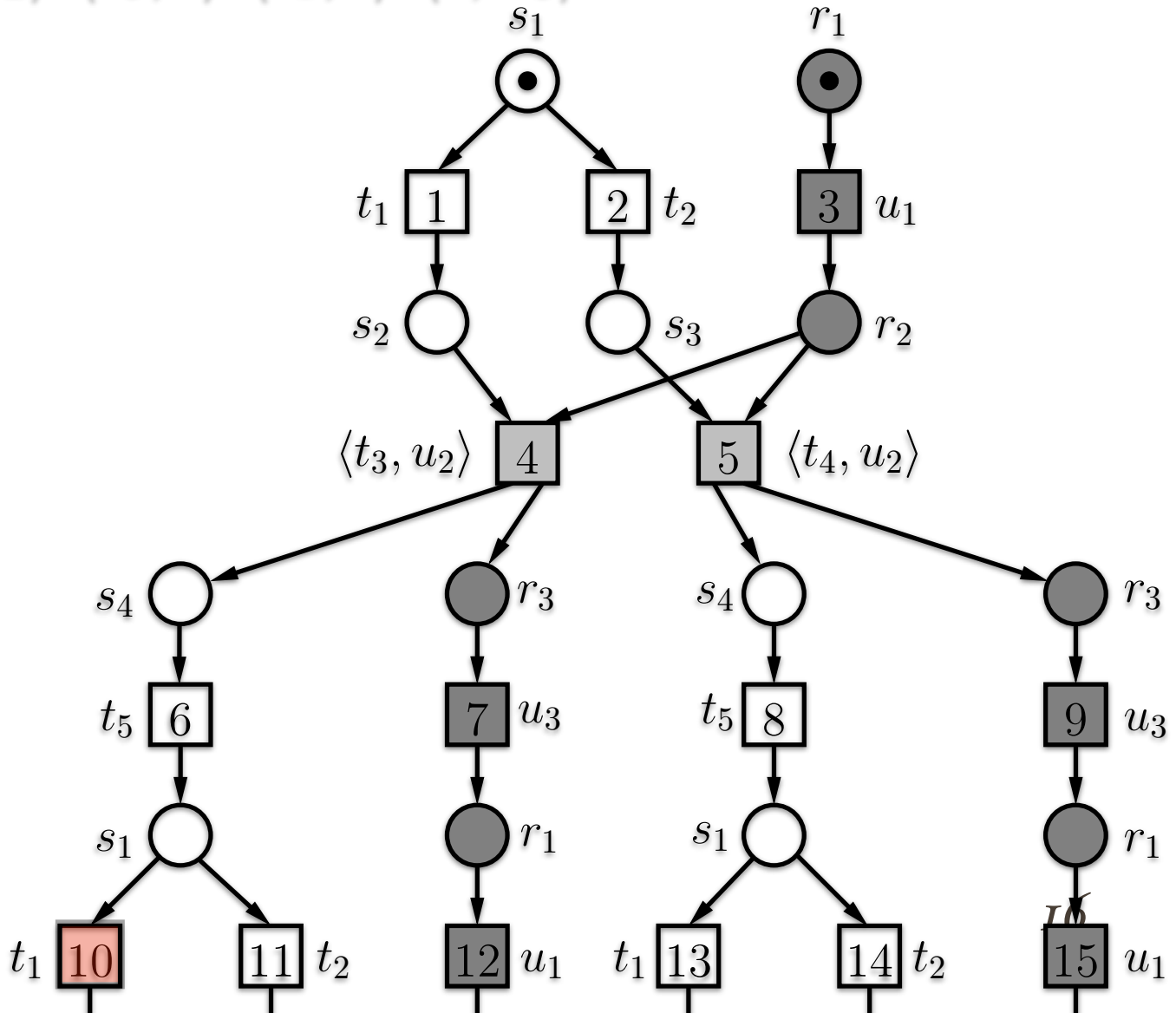
We fix a product $\mathbf{A} = \langle \mathcal{A}_1, \dots, \mathcal{A}_n, \mathbf{T} \rangle$ of transition systems, where $\mathcal{A}_i = \langle S_i, T_i, \alpha_i, \beta_i, is_i \rangle$, and a set of goal global transitions $\mathbf{G} \subseteq \mathbf{T}$. We wish to solve the problem of whether some global history of \mathbf{A} executes some transition of \mathbf{G} . Our goal is to generalize Def. 4.6 to a search scheme for an arbitrary product of transition systems.

In this section we generalize the notion of a search strategy to products. Recall that, intuitively, a search strategy determines the order in which new events are added when constructing the unfolding. In the transition system case we modeled strategies as order relations on T^* . This was possible because an event was uniquely determined by its history, and so an order on T^* induced an order on events. However, in the case of products, an event does not have a unique history, as illustrated by the example below.

Example 4.14. Consider event 10 in the unfolding of Fig. 3.3 on p. 17. Recall that this is the unfolding of the product of Fig. 2.2 on p. 7. Many occurrence sequences of events contain this event, and all of them correspond to histories of the product. Here are some examples:

Event sequence	History
1 3 4 7 6 12 10	$\langle t_1, \epsilon \rangle \langle \epsilon, u_1 \rangle \langle t_3, u_2 \rangle \langle \epsilon, u_3 \rangle \langle t_5, \epsilon \rangle \langle \epsilon, u_1 \rangle \langle t_1, \epsilon \rangle$
1 3 4 6 10	$\langle t_1, \epsilon \rangle \langle \epsilon, u_1 \rangle \langle t_3, u_2 \rangle \langle t_5, \epsilon \rangle \langle t_1, \epsilon \rangle$
3 1 4 6 10	$\langle \epsilon, u_1 \rangle \langle t_1, \epsilon \rangle \langle t_3, u_2 \rangle \langle t_5, \epsilon \rangle \langle t_1, \epsilon \rangle$
1 3 4 6 10 7	$\langle t_1, \epsilon \rangle \langle \epsilon, u_1 \rangle \langle t_3, u_2 \rangle \langle t_5, \epsilon \rangle \langle t_1, \epsilon \rangle \langle \epsilon, u_3 \rangle$

Mazurkiewicz Traces



The definition of Mazurkiewicz traces is based on the notion of independence of transitions. Recall that a component \mathcal{A}_i of \mathbf{A} participates in the execution of a global transition $\mathbf{t} = \langle t_1, \dots, t_n \rangle$ when $t_i \neq \epsilon$. We define:

Definition 4.15. *Two global transitions are independent if no component \mathcal{A}_i of \mathbf{A} participates in both of them.*

Example 4.16. The transitions $\langle t_1, \epsilon \rangle$ and $\langle \epsilon, u_1 \rangle$ of the product of Fig. 2.2 on p. 7 are independent, but the transitions $\langle t_1, \epsilon \rangle$ and $\langle t_4, u_2 \rangle$ are not, because \mathcal{A}_1 participates in both of them.

It follows easily from this definition that a pair \mathbf{t} and \mathbf{u} of independent transitions satisfies the following two properties for every $\mathbf{w}, \mathbf{w}' \in \mathbf{T}^*$:

- (1) if $\mathbf{w t u w}'$ is a history of \mathbf{A} , then so is $\mathbf{w u t w}'$; and
- (2) if $\mathbf{w t}$ and $\mathbf{w u}$ are histories of \mathbf{A} , then so are $\mathbf{w t u}$ and $\mathbf{w u t}$.

The independence relation induces an equivalence relation on the set \mathbf{T}^* of transition words. Loosely speaking, two words are equivalent if one can be obtained from the other by swapping consecutive independent transitions.

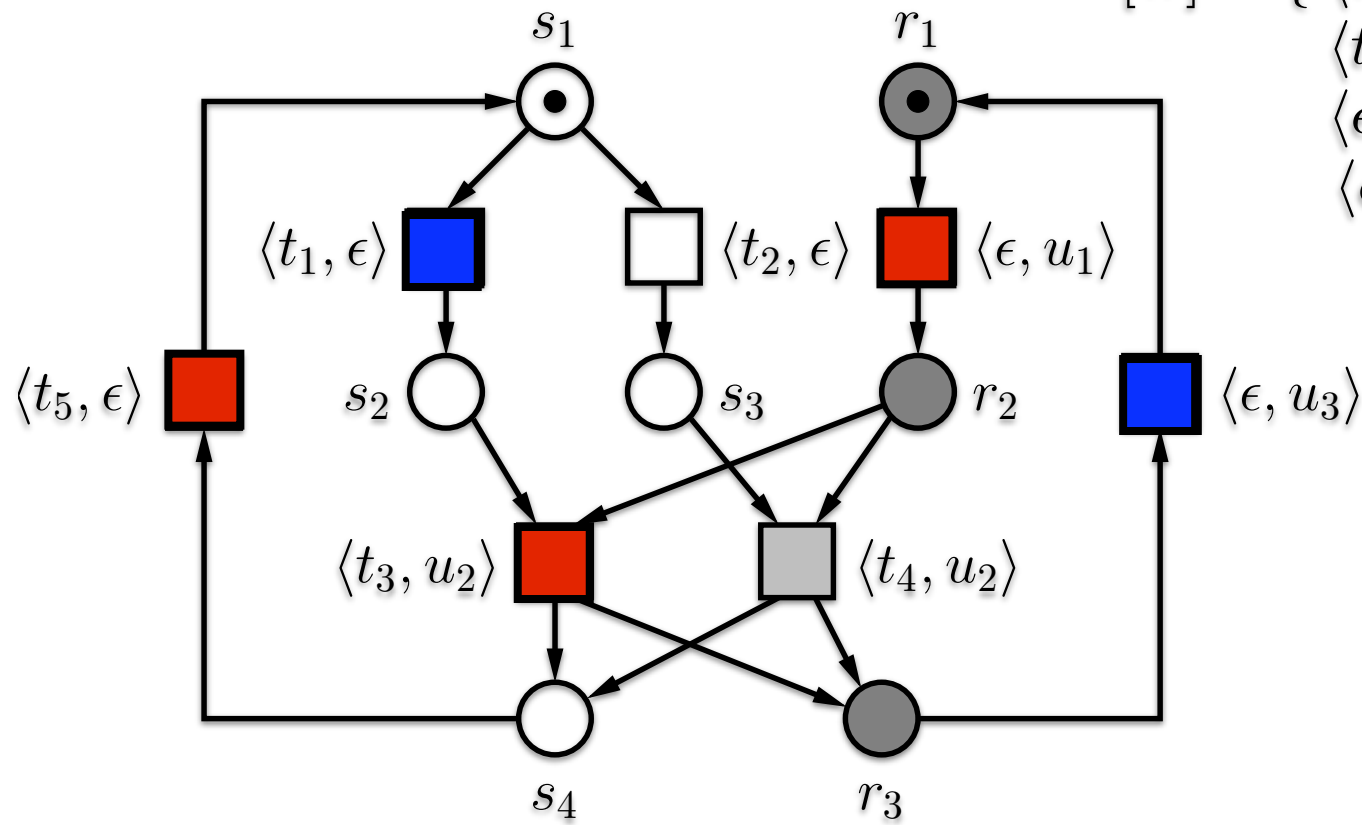
Definition 4.18. *Two transition words $\mathbf{w}, \mathbf{w}' \in \mathbf{T}^*$ are 1-equivalent, denoted by $\mathbf{w} \equiv_1 \mathbf{w}'$, if $\mathbf{w} = \mathbf{w}'$ or if there are two independent transitions \mathbf{t} and \mathbf{u} and two words $\mathbf{w}_1, \mathbf{w}_2 \in \mathbf{T}^*$ such that $\mathbf{w} = \mathbf{w}_1 \mathbf{t} \mathbf{u} \mathbf{w}_2$ and $\mathbf{w}' = \mathbf{w}_1 \mathbf{u} \mathbf{t} \mathbf{w}_2$. Two words $\mathbf{w}, \mathbf{w}' \in \mathbf{T}^*$ are equivalent if $\mathbf{w} \equiv \mathbf{w}'$, where \equiv denotes the transitive closure of \equiv_1 .*

Since \equiv_1 is reflexive and symmetric, \equiv is an equivalence relation. It follows easily from this definition and property (1) above that if \mathbf{h} is a history of \mathbf{A} , then every word $\mathbf{w} \equiv \mathbf{h}$ is also a history of \mathbf{A} .

Definition 4.19. *A Mazurkiewicz trace (or just a trace) of a product \mathbf{A} is an equivalence class of the relation \equiv . The trace of a word \mathbf{w} is denoted by $[\mathbf{w}]$, and the set of all traces of \mathbf{A} by $[\mathbf{T}^*]$.*

A trace of \mathbf{A} is a history trace if all its elements are histories.

Example 4.20. The sequence $\mathbf{w} = \langle t_1, \epsilon \rangle \langle \epsilon, u_1 \rangle \langle t_3, u_2 \rangle \langle t_5, \epsilon \rangle \langle \epsilon, u_3 \rangle$ is a history of the product of Fig. 2.2 on p. 7. Its corresponding history trace is:



$$[\mathbf{w}] = \left\{ \begin{array}{l} \langle t_1, \epsilon \rangle \langle \epsilon, u_1 \rangle \langle t_3, u_2 \rangle \langle t_5, \epsilon \rangle \langle \epsilon, u_3 \rangle, \\ \langle t_1, \epsilon \rangle \langle \epsilon, u_1 \rangle \langle t_3, u_2 \rangle \langle \epsilon, u_3 \rangle \langle t_5, \epsilon \rangle, \\ \langle \epsilon, u_1 \rangle \langle t_1, \epsilon \rangle \langle t_3, u_2 \rangle \langle t_5, \epsilon \rangle \langle \epsilon, u_3 \rangle, \\ \langle \epsilon, u_1 \rangle \langle t_1, \epsilon \rangle \langle t_3, u_2 \rangle \langle \epsilon, u_3 \rangle \langle t_5, \epsilon \rangle \end{array} \right\}.$$

$\mathbf{w}' = \langle t_1, \epsilon \rangle \langle \epsilon, u_3 \rangle \langle t_1, \epsilon \rangle$ is not a history.

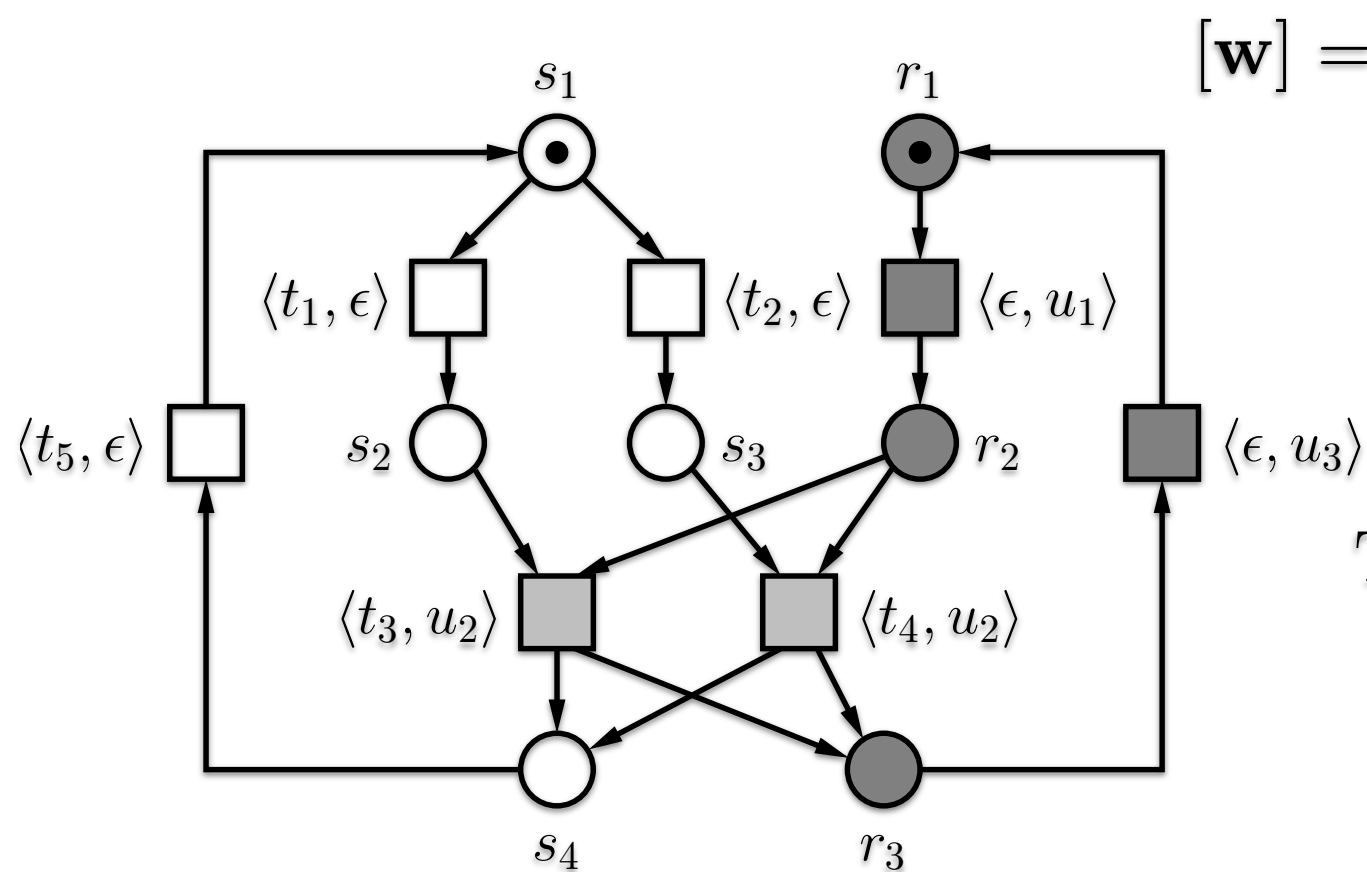
$$[\mathbf{w}'] = \left\{ \begin{array}{l} \langle t_1, \epsilon \rangle \langle \epsilon, u_3 \rangle \langle t_1, \epsilon \rangle, \\ \langle \epsilon, u_3 \rangle \langle t_1, \epsilon \rangle \langle t_1, \epsilon \rangle, \\ \langle t_1, \epsilon \rangle \langle t_1, \epsilon \rangle \langle \epsilon, u_3 \rangle \end{array} \right\}.$$

Theorem 4.21. For every $i \in \{1, \dots, n\}$, let $\mathbf{T}_i \subseteq \mathbf{T}$ be the set of global transitions of \mathbf{A} in which \mathcal{A}_i participates. Two words $\mathbf{w}, \mathbf{w}' \in \mathbf{T}^*$ satisfy $\mathbf{w} \equiv \mathbf{w}'$ if and only if for every $i \in \{1, \dots, n\}$ their projections onto \mathbf{T}_i coincide.

Example 4.22. Consider the trace $[\mathbf{w}]$ of Ex. 4.20. In this case we have

$$\mathbf{T}_1 = \{ \langle t_1, \epsilon \rangle, \langle t_2, \epsilon \rangle, \langle t_3, u_2 \rangle, \langle t_4, u_2 \rangle, \langle t_5, \epsilon \rangle \}, \text{ and}$$

$$\mathbf{T}_2 = \{ \langle t_3, u_2 \rangle, \langle t_4, u_2 \rangle, \langle \epsilon, u_1 \rangle, \langle \epsilon, u_3 \rangle \}.$$



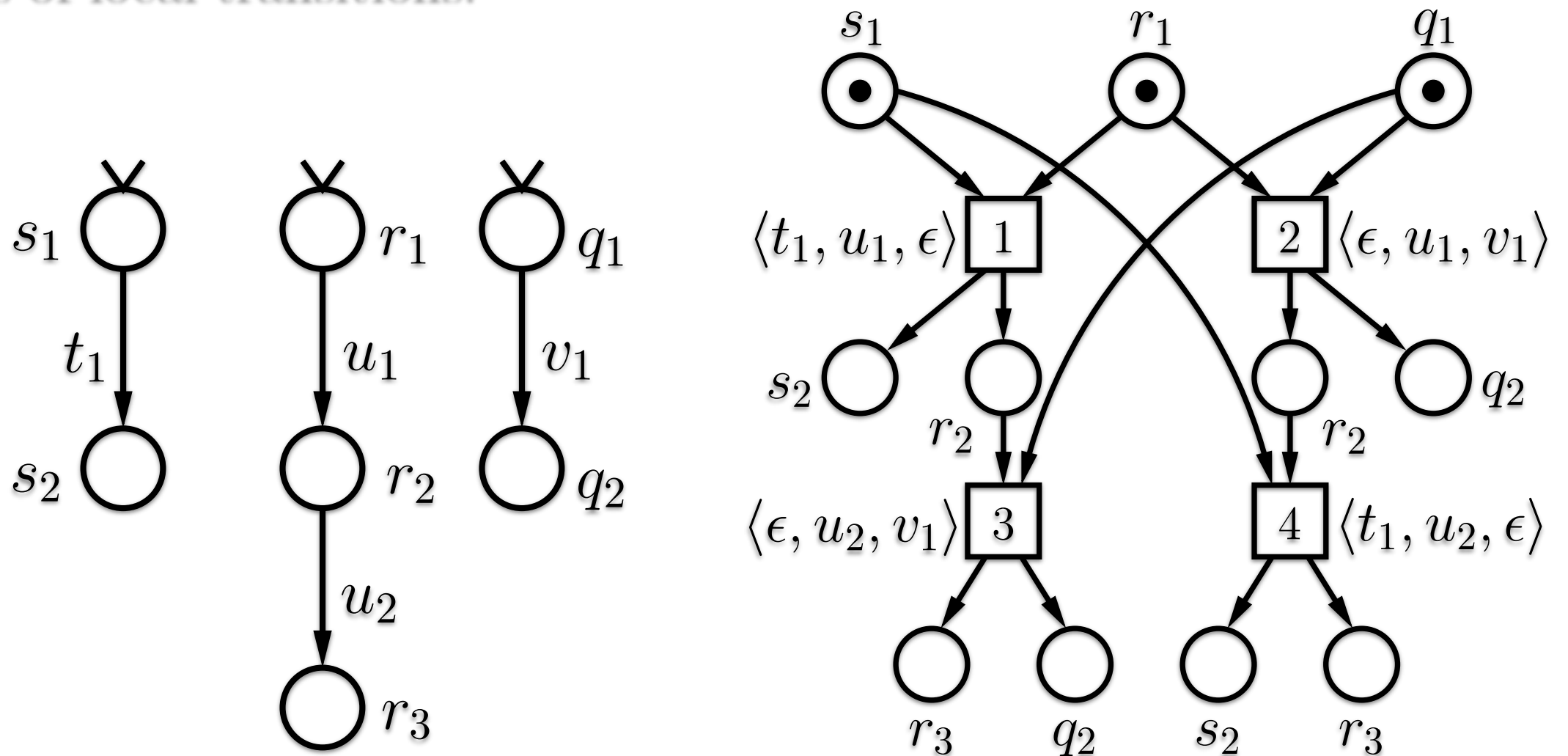
$$[\mathbf{w}] = \left\{ \begin{array}{ccccc} \langle t_1, \epsilon \rangle & \langle \epsilon, u_1 \rangle & \langle t_3, u_2 \rangle & \langle t_5, \epsilon \rangle & \langle \epsilon, u_3 \rangle, \\ \langle t_1, \epsilon \rangle & \langle \epsilon, u_1 \rangle & \langle t_3, u_2 \rangle & \langle \epsilon, u_3 \rangle & \langle t_5, \epsilon \rangle, \\ \langle \epsilon, u_1 \rangle & \langle t_1, \epsilon \rangle & \langle t_3, u_2 \rangle & \langle t_5, \epsilon \rangle & \langle \epsilon, u_3 \rangle, \\ \langle \epsilon, u_1 \rangle & \langle t_1, \epsilon \rangle & \langle t_3, u_2 \rangle & \langle \epsilon, u_3 \rangle & \langle t_5, \epsilon \rangle \end{array} \right\}.$$

The projections are

$$\begin{array}{ccc} \langle t_1, \epsilon \rangle & \langle t_3, u_2 \rangle & \langle t_5, \epsilon \rangle \\ \langle \epsilon, u_1 \rangle & \langle t_3, u_2 \rangle & \langle \epsilon, u_3 \rangle. \end{array}$$

At this point the reader might like to ask the following question: Does Thm. 4.21 still hold if we replace the projections of \mathbf{w} and \mathbf{w}' onto $\mathbf{T}_1, \dots, \mathbf{T}_n$ by their projections onto T_1, \dots, T_n , the sets of *local* transitions of $\mathcal{A}_1, \dots, \mathcal{A}_n$? This would look more natural, since then the projections of a global history would be local histories of its components. However, the theorem then fails, as shown by the following example.

Example 4.23. Consider the product shown in Fig. 4.5(a), and the global histories $\mathbf{w}_1 = \langle t_1, u_1, \epsilon \rangle \langle \epsilon, u_2, v_1 \rangle$ and $\mathbf{w}_2 = \langle \epsilon, u_1, v_1 \rangle \langle t_1, u_2, \epsilon \rangle$. We have $[\mathbf{w}_1] = \{\mathbf{w}_1\} \neq \{\mathbf{w}_2\} = [\mathbf{w}_2]$. However, the projections of \mathbf{w}_1 and \mathbf{w}_2 onto the sets T_1, T_2 , and T_3 of local transitions coincide; they are in both cases t_1, u_1u_2 , and v_1 . So a trace is not characterized by the projections of its elements onto the sets of local transitions.



$$\mathbf{T} = \{ \langle t_1, u_1, \epsilon \rangle, \langle \epsilon, u_1, v_1 \rangle, \langle t_1, u_2, \epsilon \rangle, \langle \epsilon, u_2, v_1 \rangle \}$$

4.3.2 Search Strategies as Orders on Mazurkiewicz Traces

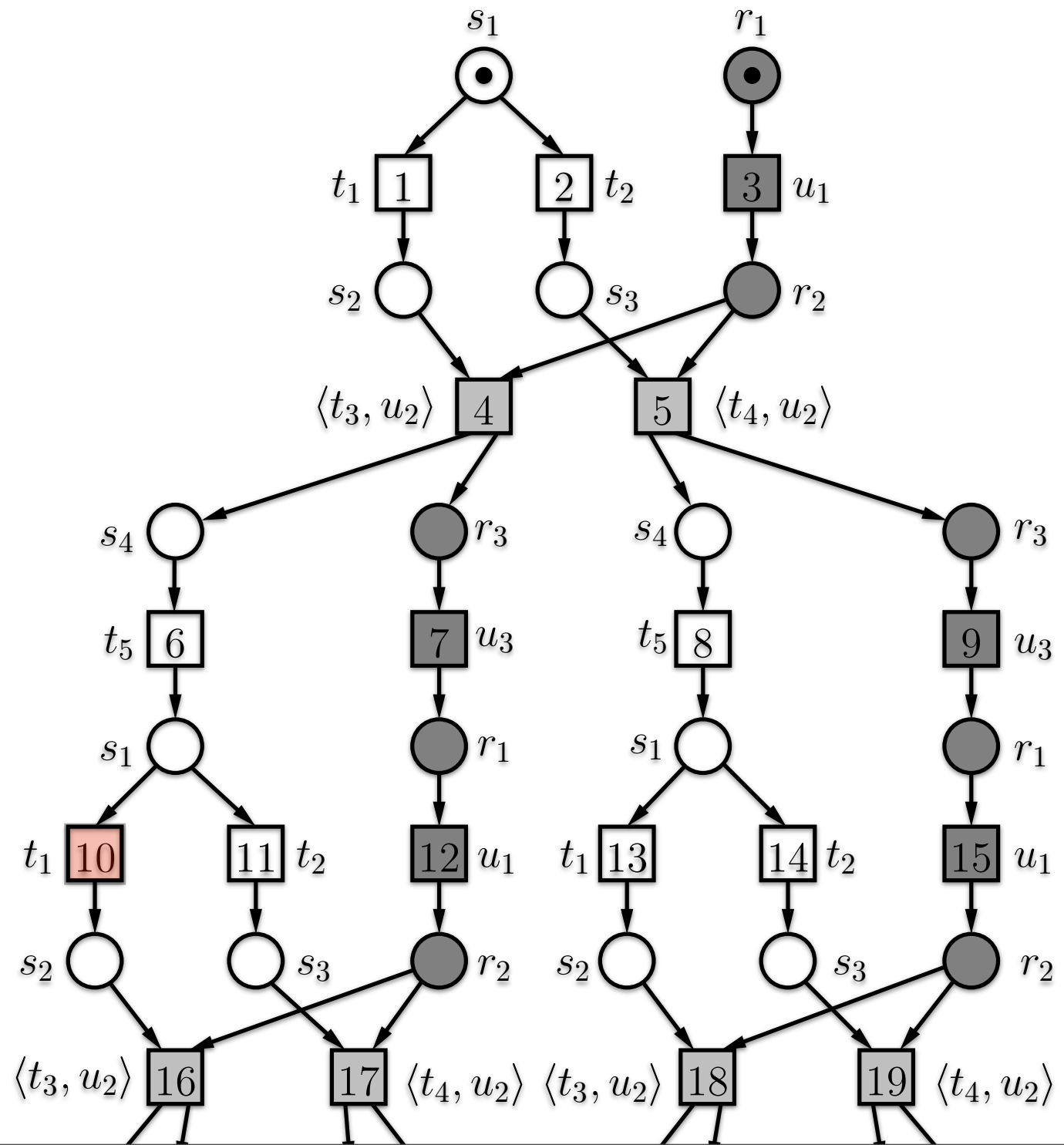
We define search strategies following the same steps of the transition system case (see Sect. 4.1). First, we formally define the set of histories of an event (Def. 4.26). Then we show that this set is always a Mazurkiewicz trace (Prop. 4.28). So an order on Mazurkiewicz traces induces an order on events, and so it makes sense to define strategies as orders on Mazurkiewicz traces (Def. 4.32).

We start by introducing the notion of the past of an event. Intuitively, this is the set of events that must occur for the event to occur.

Definition 4.24. *The past of an event e , denoted by $\text{past}(e)$, is the set of events e' such that $e' \leq e$ (recall that $e' < e$ if there is a path leading from e' to e).²*

It is easy to see that $\text{past}(e)$ is always a configuration.

Example 4.25. The past of event 10 in Fig. 3.3 on p. 17 is $past(10) = \{1, 3, 4, 6, 10\}$. While the past of an event is a configuration, not every configuration is the past of an event. For instance the configuration $\{2, 3\}$ is not the past of an event. However, it is easy to see that every configuration is the union of the pasts of some events. For instance, for $\{2, 3\}$ we have $past(2) = \{2\}$ and $past(3) = \{3\}$.



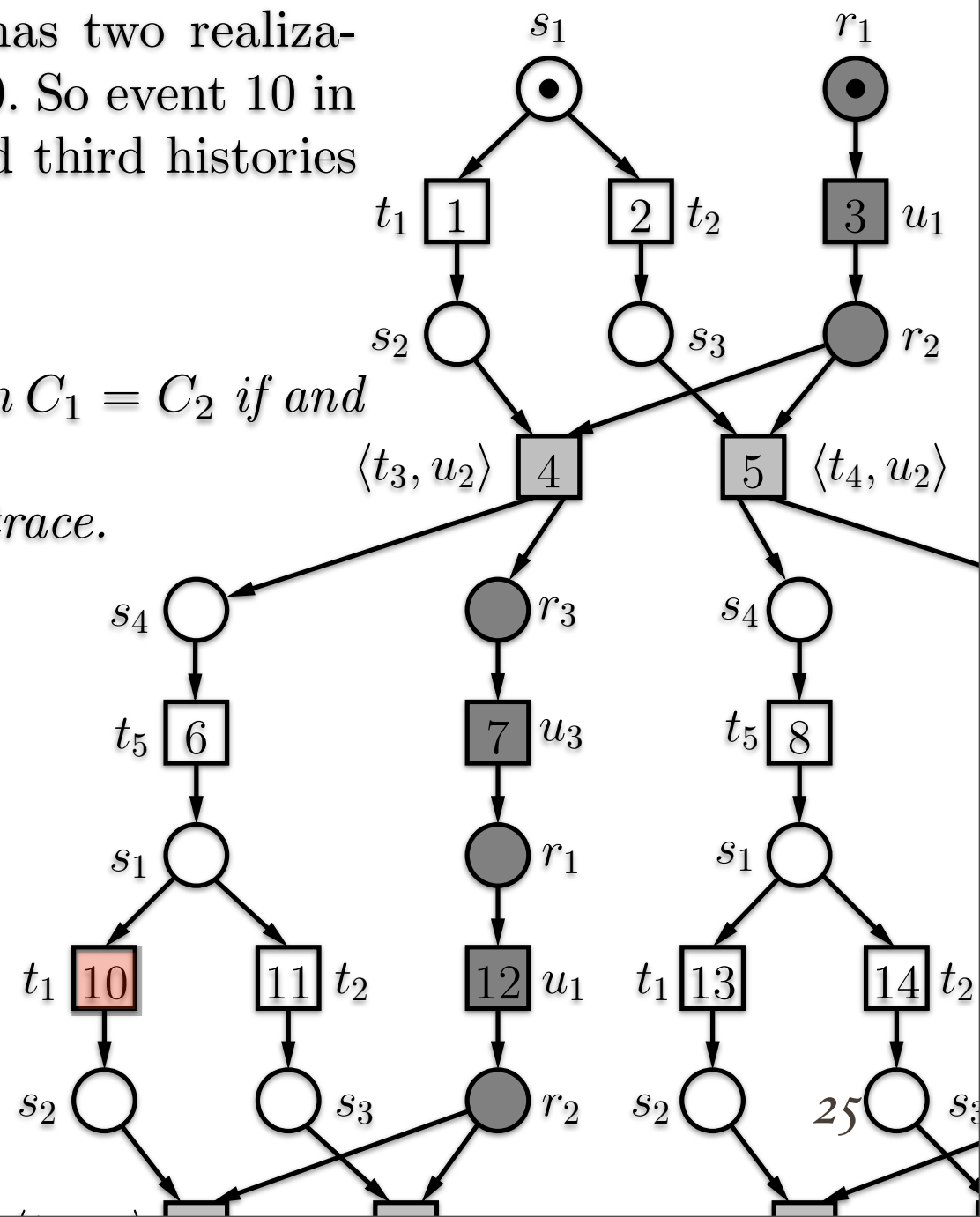
Now we define the set of histories of an event as the set of realizations of its past.

Definition 4.26. A transition word $t_1 t_2 \dots t_n$ is a **history** of a configuration C if there is a realization $e_1 e_2 \dots e_n$ of C such that e_i is labeled by t_i for every $i \in \{1, \dots, n\}$. The set of histories of C is denoted by $\mathbf{H}(C)$. If $C = \text{past}(e)$ for some event e , then we also call the elements of $\mathbf{H}(\text{past}(e))$ histories of e . To simplify notation, we write $\mathbf{H}(e)$ instead of $\mathbf{H}(\text{past}(e))$.

Example 4.27. The configuration $\text{past}(10) = \{1, 3, 4, 6, 10\}$ has two realizations, namely the occurrence sequences 1 3 4 6 10 and 3 1 4 6 10. So event 10 in Fig. 3.3 on p. 17 has two histories, which are the second and third histories in the list of four shown in Ex. 4.14 on p. 48.

Proposition 4.28. (a) Let C_1, C_2 be two configurations. Then $C_1 = C_2$ if and only if $\mathbf{H}(C_1) = \mathbf{H}(C_2)$.
 (b) Let C be a configuration. Then $\mathbf{H}(C)$ is a Mazurkiewicz trace.

Event sequence
1 3 4 7 6 12 10
1 3 4 6 10
3 1 4 6 10
1 3 4 6 10 7



Proposition 4.28 shows that we can define strategies for products as orders on the Mazurkiewicz traces of \mathbf{A} . Recall however that in the transition system case a strategy was defined as an order that refines the prefix order, the idea being that a history must be generated before any of its extensions are generated. We need the same condition in the general case.

Definition 4.30. *The concatenation of two traces $[\mathbf{w}]$ and $[\mathbf{w}']$ is denoted by $[\mathbf{w}] [\mathbf{w}']$ and defined as the trace $[\mathbf{w} \mathbf{w}']$. We say that $[\mathbf{w}]$ is a prefix of $[\mathbf{w}']$ if there is a trace $[\mathbf{w}'']$ such that $[\mathbf{w}'] = [\mathbf{w}] [\mathbf{w}']$.*

We list some useful properties of trace concatenation and prefixes.

Proposition 4.31.

- (a) *The prefix relation on traces is an order.*
- (b) $[\mathbf{w}_1][\mathbf{w}_2] \supseteq \{\mathbf{v}_1 \mathbf{v}_2 \mid \mathbf{v}_1 \in [\mathbf{w}_1], \mathbf{v}_2 \in [\mathbf{w}_2]\}$.
- (c) *If $[\mathbf{w}_1] = [\mathbf{w}_2]$ then $[\mathbf{w}][\mathbf{w}_1][\mathbf{w}'] = [\mathbf{w}][\mathbf{w}_2][\mathbf{w}']$ for all words \mathbf{w} and \mathbf{w}' .*
- (d) *If $[\mathbf{w}][\mathbf{w}_1][\mathbf{w}'] = [\mathbf{w}][\mathbf{w}_2][\mathbf{w}']$ for some words \mathbf{w} and \mathbf{w}' , then $[\mathbf{w}_1] = [\mathbf{w}_2]$.*

We can now formulate the generalization of search strategies.

Definition 4.32. *A search strategy for \mathbf{A} is an order on $[\mathbf{T}^*]$ that refines the prefix order on traces.*

research scheme for transition systems

Definition 4.6. Let \prec be a search strategy. An event e is feasible if no event $e' \prec e$ is a terminal. A feasible event e is a terminal if either

- (a) it is labeled with a transition of G , or
- (b) there is a feasible event $e' \prec e$, called the companion of e , such that $St(e') = St(e)$.

A terminal is successful if it is of type (a). The \prec -final prefix is the prefix of the unfolding of A containing the feasible events.

research scheme for products

Definition 4.35. Let \prec be a search strategy on $[\mathbf{T}^*]$. An event e of the unfolding of \mathbf{A} is feasible if no event $e' \prec e$ is a terminal. A feasible event e is a terminal if either

- (a) e is labeled with a transition of \mathbf{G} , or
- (b) there is a feasible event $e' \prec e$, called the companion of e , such that $\mathbf{St}(e') = \mathbf{St}(e)$.

A terminal is successful if it is of type (a). The \prec -final prefix is the prefix of the unfolding of \mathbf{A} containing the feasible events.


$$\mathbf{St}(e) = \mathbf{St}(\text{past}(e))$$

the global state reached

It is easy to show that the scheme is well-defined and sound for every strategy (compare with Lemma 4.8, Prop. 4.9 and Prop. 4.10).

Lemma 4.36. *Let \prec be an arbitrary search strategy, and let (F, T) be a pair of sets of events satisfying the conditions of Def. 4.35 for the sets of feasible and terminal events, respectively. For every event $e \in F$, every history of $\mathbf{H}(e)$ has length at most $nK + 1$, where n is the number of components of \mathbf{A} and K is the number of reachable global states of \mathbf{A} .*

Proof. Assume that some history of $\mathbf{H}(e)$ has length greater than $nK + 1$. Then, by the pigeonhole principle there is a component \mathcal{A}_i of \mathbf{A} and two i -events $e_1 < e_2 < e$ such that $\mathbf{St}_i(e_1) = \mathbf{St}_i(e_2)$. Since $e_1 < e_2$ and the search strategy \prec refines the prefix order, we have $e_1 \prec e_2$. By condition (b) we have $e_2 \in T$. Since $e_2 < e$, e cannot be feasible, i.e., $e \notin F$, and we are done. \square

A direct corollary of the proof above is that for all non-terminal feasible events e the maximal length of the histories in $\mathbf{H}(e)$ is at most nK .

Proposition 4.37. *The search scheme of Def. 4.35 is well-defined for every search strategy \prec , i.e., there is a unique set of feasible events and a unique set of terminal events satisfying the conditions of the definition. Moreover, the \prec -final prefix is finite.*

Proof. Analogous to the proof of Prop. 4.9. □

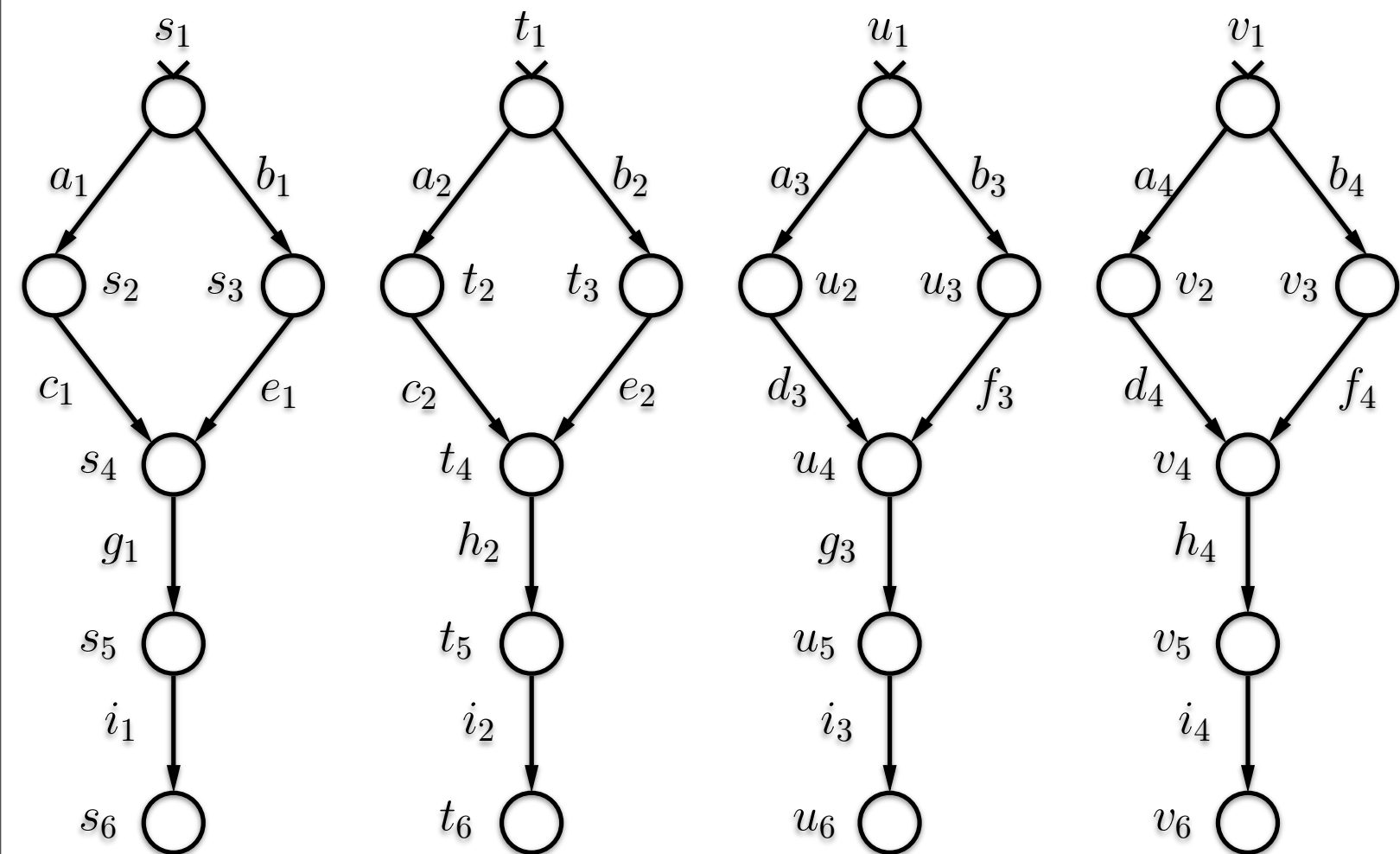
Proposition 4.38. *The search scheme of Def. 4.35 is sound for every strategy.*

In the worst case, the size of the final prefix can be exponential in the number K of reachable global states of \mathbf{A} . This is not surprising, since this is already the case for transition systems, which are products with only one component. As in the case of transition systems, we can do better by using *total* strategies.

Theorem 4.39. *If \prec is a total search strategy on $[\mathbf{T}^*]$, then the \prec -final prefix generated by the search scheme of Def. 4.35 with \prec as search strategy has at most K non-terminal events.*

Proof. Analogous to the proof of Thm. 4.13. □

Unfortunately, a direct generalization of Thm. 4.11 *does not hold*: the search scheme of Def. 4.35 is **not complete** for every strategy, as shown by the next example.



$$\mathbf{T} = \{ \mathbf{a} = \langle a_1, a_2, a_3, a_4 \rangle, \mathbf{b} = \langle b_1, b_2, b_3, b_4 \rangle, \mathbf{c} = \langle c_1, c_2, \epsilon, \epsilon \rangle, \\ \mathbf{d} = \langle \epsilon, \epsilon, d_3, d_4 \rangle, \mathbf{e} = \langle e_1, e_2, \epsilon, \epsilon \rangle, \mathbf{f} = \langle \epsilon, \epsilon, f_3, f_4 \rangle, \\ \mathbf{g} = \langle g_1, \epsilon, g_3, \epsilon \rangle, \mathbf{h} = \langle \epsilon, h_2, \epsilon, h_4 \rangle, \mathbf{i} = \langle i_1, i_2, i_3, i_4 \rangle \}$$

$$\mathbf{G} = \{ \mathbf{i} \}$$

Fig. 4.6. An instance of the executability problem

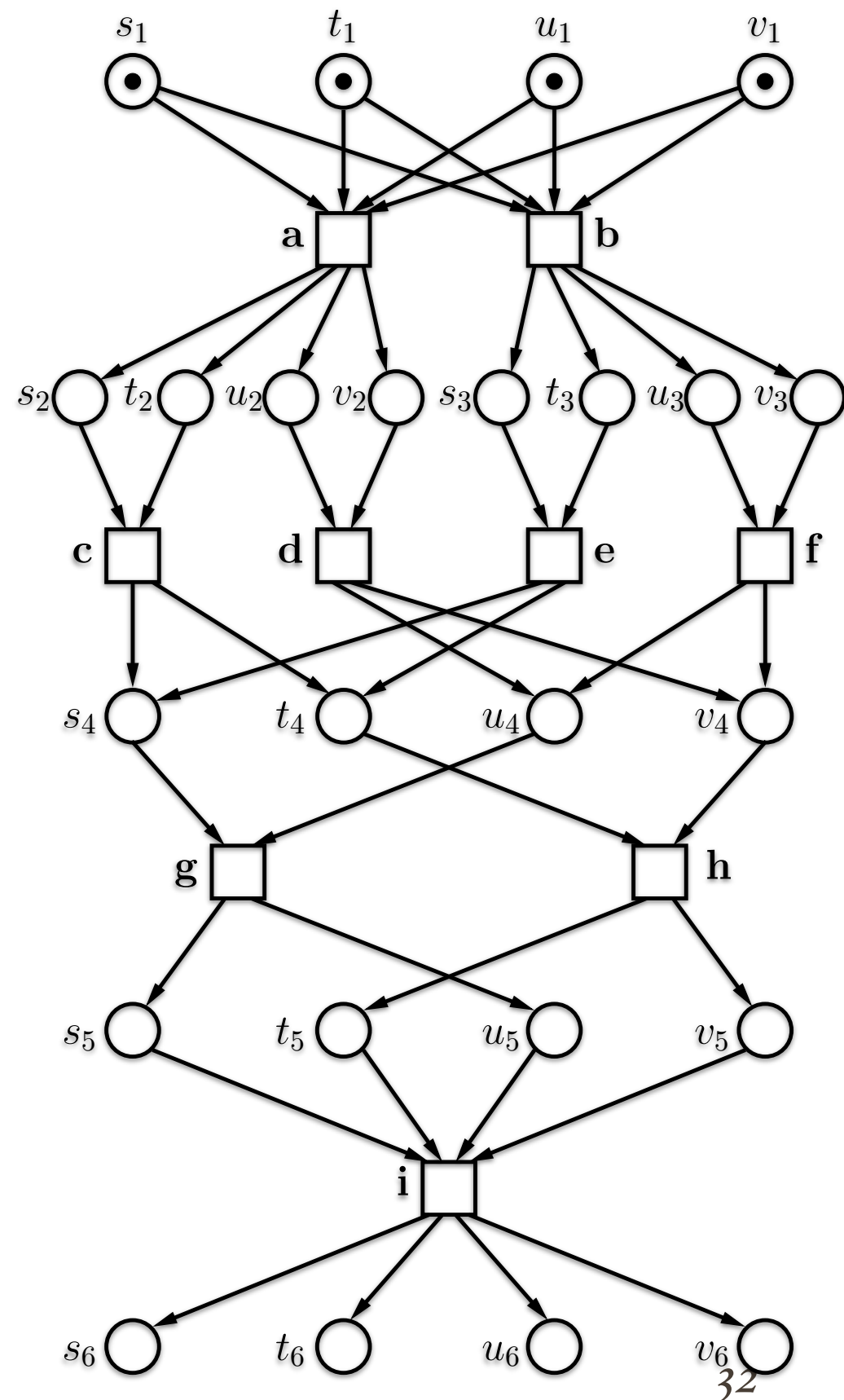


Fig. 4.7. Petri net representation of the product of Fig. 4.6

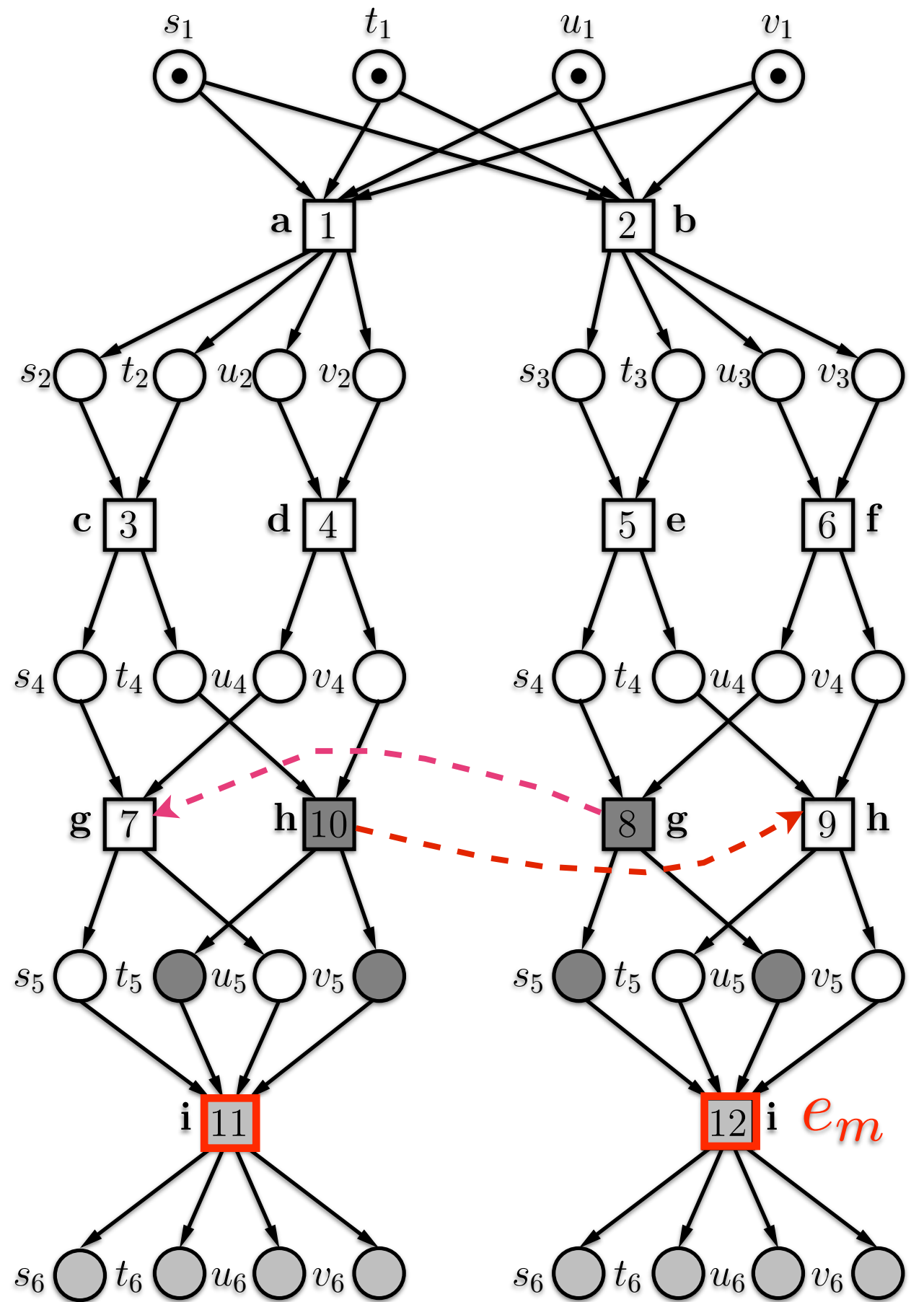
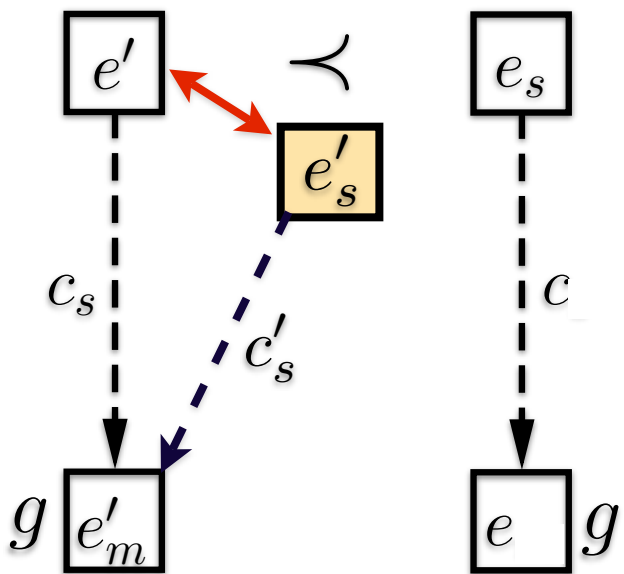


Fig. 4.8. Unfolding of the product of Fig. 4.6



Contradiction. Since $H(e'_s)c'_s = H(e'_m) = H(e')c_s$, we have $e' < e'_m$ and $e'_s < e'_m$. So there are **three possible cases:**

- **$e'_s < e'$.** Then, since e'_s is a spoiler and spoilers are terminals, e' is not feasible, contradicting our assumption that e' is the companion of e_s , which according to Def. 4.6 requires e' to be feasible.
- **$e'_s = e'$.** Then, since $H(e'_s)c'_s = H(e'_m) = H(e')c_s$, we have $c_s = c'_s$. Moreover, since $e'_s = e'$ and $e' \prec e_s$ we have $e'_s \prec e_s$. This implies $e'_m \ll e_m$, contradicting the minimality of e_m .
- **$e' < e'_s$.** Then, since $H(e'_s)c'_s = H(e'_m) = H(e')c_s$, the computation c'_s is shorter than c_s , and so $e'_m \ll e_m$, contradicting the minimality of e_m .

□

The next example shows that the size of the final prefix depends on the choice of strategy. In the worst case, the final prefix can be exponentially larger than the transition system.

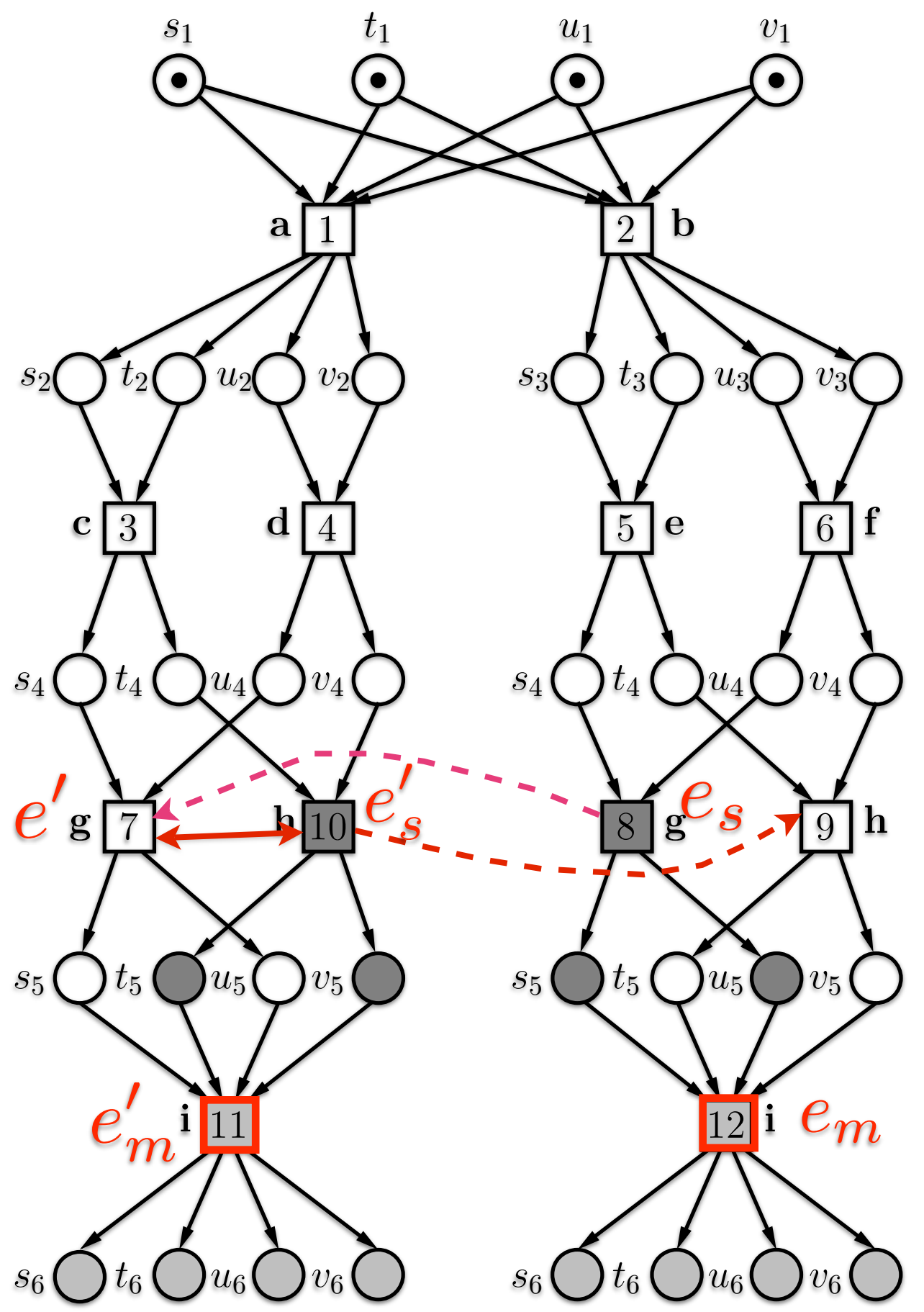
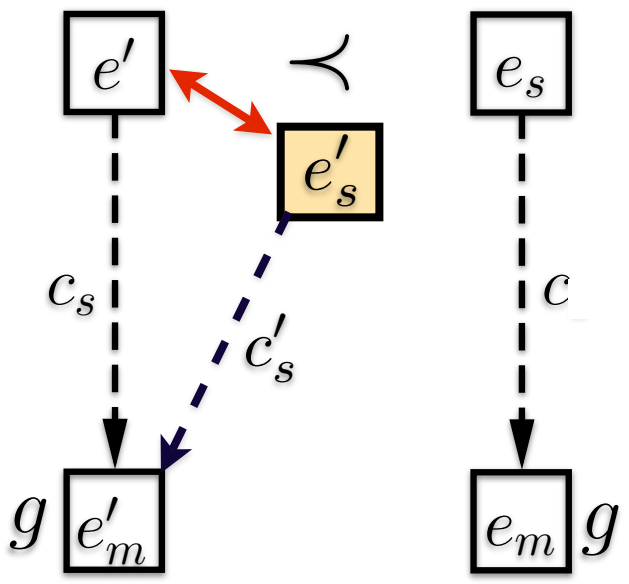


Fig. 4.8. Unfolding of the product of Fig. 4.6 35

The question is, which are the search strategies that in combination with the search scheme lead to complete search procedures? The next definition introduces such a class of strategies.

Definition 4.41. *A strategy \prec on $[\mathbf{T}^*]$ is adequate if*

- *it is well-founded, and*
- *it is preserved by extensions: For all traces $[\mathbf{w}], [\mathbf{w}'], [\mathbf{w}''] \in [\mathbf{T}^*]$, if $[\mathbf{w}] \prec [\mathbf{w}']$, then $[\mathbf{w}] [\mathbf{w}''] \prec [\mathbf{w}'] [\mathbf{w}'']$.*

The following surprising result is due to Chatain and Khomenko. The proof requires some notions of the theory of well-quasi-orders, and can be found in Sect. 5.4 of the next chapter.

Theorem 4.43. *The search scheme of Def. 4.35 is complete for all adequate strategies.*

Proof. Let \prec be an adequate search strategy. Assume that some goal transition $\mathbf{g} \in \mathbf{G}$ is executable, but no terminal of the final prefix is successful. We derive a contradiction.

We follow the scheme of the completeness proof of Thm. 4.11, just changing the definition of minimal witness, and using the definition of an adequate strategy to derive the contradiction.

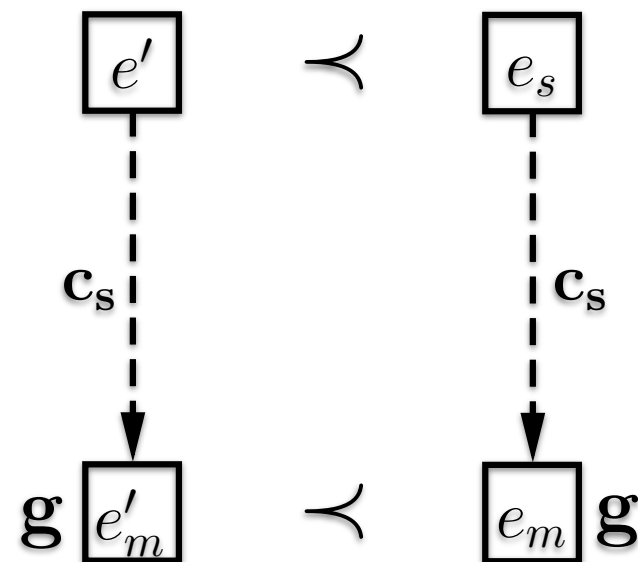
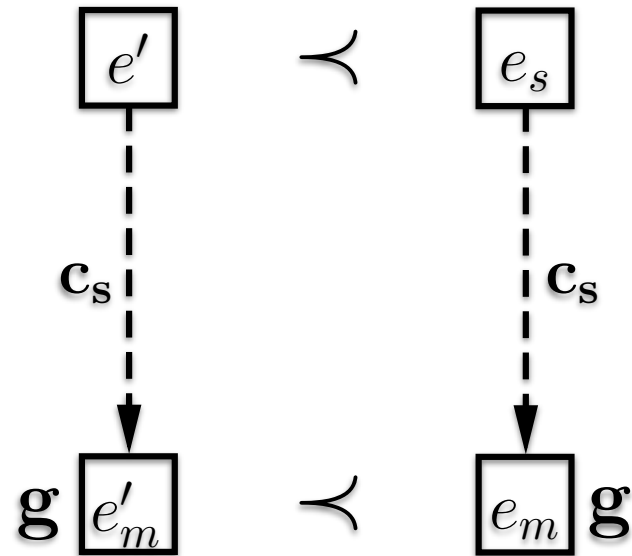


Fig. 4.9. Illustration of the proof of Thm. 4.43



Witnesses. Let an event of the unfolding of \mathbf{A} be a *witness* if it is labeled with \mathbf{g} . Using the same argument as in the proof of Thm. 4.11, we conclude that for every witness e there is an unsuccessful terminal $e_s < e$ that we call the *spoiler* of e .

Minimal witnesses. Since \prec is well-founded by definition, the set of witnesses has at least one minimal element e_m w.r.t. \prec . Let e_s be the spoiler of e_m (see Fig. 4.9), and let $[\mathbf{c}_s]$ be a trace satisfying $\mathbf{H}(e_m) = \mathbf{H}(e_s)[\mathbf{c}_s]$. Since e_s is an unsuccessful terminal, it has a companion $e' \prec e_s$ such that $\mathbf{St}(e') = \mathbf{St}(e_s)$, and so $\mathbf{H}(e')[\mathbf{c}_s]$ is also a history of \mathbf{A} . Let e'_m be the event satisfying $\mathbf{H}(e'_m) = \mathbf{H}(e')[\mathbf{c}_s]$. Then e'_m is labeled with \mathbf{g} , and so it is a witness.

Contradiction. Since \prec is preserved by extensions and $\mathbf{H}(e') \prec \mathbf{H}(e_s)$ holds, we have $\mathbf{H}(e'_m) = \mathbf{H}(e')[\mathbf{c}_s] \prec \mathbf{H}(e_s)[\mathbf{c}_s] = \mathbf{H}(e_m)$, and so $\mathbf{H}(e'_m) \prec \mathbf{H}(e_m)$. But this implies $e'_m \prec e_m$, which contradicts the minimality of e_m . \square

Let us summarize the results of this section and Sect. 4.4. The search scheme of Def. 4.35 is well-defined and sound for all search strategies. Moreover,

- the final prefix has at most K non-terminal events (where K is the number of reachable global states) for total strategies, and
- the scheme is complete for adequate strategies.

The obvious question is whether total *and* adequate strategies exist. In the rest of the section we show that this is indeed the case.

In order to avoid confusions, we use the following notational convention:

Notation 2. *We denote strategies by adding a mnemonic subscript to the symbol \prec , as in \prec_x . Given a strategy \prec_x , we write $[\mathbf{w}] =_x [\mathbf{w}']$ to denote that neither $[\mathbf{w}] \prec_x [\mathbf{w}']$ nor $[\mathbf{w}'] \prec_x [\mathbf{w}]$ holds.*

A simple example of an adequate strategy is the size strategy.

Definition 4.44. *The size strategy \prec_s on $[\mathbf{T}^*]$ is defined as: $[\mathbf{w}] \prec_s [\mathbf{w}']$ if $|\mathbf{w}| < |\mathbf{w}'|$, i.e., if \mathbf{w} is shorter than \mathbf{w}' .*

Notice that the size strategy is well-defined because all words that belong to a trace have the same length. Observe also that, as required of a strategy, it refines the prefix order. Finally, the size strategy is clearly adequate, because $[\mathbf{w}] \prec_s [\mathbf{w}']$ implies $|\mathbf{w}| < |\mathbf{w}'|$ implies $|\mathbf{w}\mathbf{w}''| < |\mathbf{w}'\mathbf{w}''|$ implies $[\mathbf{w}][\mathbf{w}''] \prec_s [\mathbf{w}'][\mathbf{w}'']$ for every trace $[\mathbf{w}], [\mathbf{w}'], [\mathbf{w}'']$.

Unfortunately, as we saw in Ex. 4.12 on p. 47, the size strategy may lead to very large final prefixes, even for transition systems. In the worst case, the prefix can be exponentially larger than the transition system itself, and so potentially much too large for verification purposes.

The Parikh strategy is a refinement of the size strategy that compares not only the total number of occurrences of transitions, but also the number of occurrences of each individual transition. In order to define it, we introduce the *Parikh image* of a trace.

Definition 4.45. Let $[\mathbf{w}]$ be a trace. The Parikh mapping of $[\mathbf{w}]$, denoted by $\mathcal{P}([\mathbf{w}])$, is the mapping that assigns to each global transition \mathbf{t} the number of times that \mathbf{t} occurs in \mathbf{w} .

Notice that the Parikh mapping is well-defined because all the words of a trace have the same Parikh mapping. The Parikh strategy is defined in two stages: first, the sizes of the traces are compared, and then, if necessary, their Parikh mappings.

Definition 4.46. Let $<_a$ be a total order on \mathbf{T} , called the alphabetical order. The Parikh strategy \prec_P on $[\mathbf{T}^*]$ is defined as follows: $[\mathbf{w}] \prec_P [\mathbf{w}']$ if either $[\mathbf{w}] \prec_s [\mathbf{w}']$, or $[\mathbf{w}] =_s [\mathbf{w}']$ and there is a global transition \mathbf{t} such that

- $\mathcal{P}([\mathbf{w}])(\mathbf{t}) < \mathcal{P}([\mathbf{w}'])(\mathbf{t})$, and
- $\mathcal{P}([\mathbf{w}])(\mathbf{t}') = \mathcal{P}([\mathbf{w}'])(\mathbf{t}')$ for every $\mathbf{t}' <_a \mathbf{t}$.

The Parikh strategy refines the prefix order because the size strategy does. It is easily seen to be adequate: Since $\mathcal{P}([\mathbf{w}][\mathbf{w}']) = \mathcal{P}([\mathbf{w}]) + \mathcal{P}([\mathbf{w}'])$, we have that $[\mathbf{w}] \prec_P [\mathbf{w}']$ implies $[\mathbf{w}][\mathbf{w}'] \prec_P [\mathbf{w}'][\mathbf{w}']$ for every trace $[\mathbf{w}], [\mathbf{w}'], [\mathbf{w}']$.

The Parikh strategy leads to smaller final prefixes than the size strategy, and it is in fact a useful strategy in practical applications. However, it is easy to show that it is not a total strategy, not even for products with only one component. So Thm. 4.39 cannot be applied to it.

4.5.2 Distributed Strategies

In this section we show that the problem of finding a total adequate strategy on the set $[\mathbf{T}^*]$ of traces can be reduced to the much simpler problem of finding a total adequate strategy for the set \mathbf{T}^* of words.

Recall Thm. 4.21: Given a trace, the projections of its elements onto $\mathbf{T}_1, \dots, \mathbf{T}_n$ coincide, where \mathbf{T}_i is the set of global transitions in which the i th component participates. Moreover, these projections characterize the trace, i.e., different traces have different projections. It follows that searching for total adequate orders on Mazurkiewicz traces reduces to searching for total adequate orders on their tuples of projections onto $\mathbf{T}_1, \dots, \mathbf{T}_n$. Since these are just tuples of words, we can try the following approach: find a total adequate order on \mathbf{T}^* , and then use some generic procedure to “lift” it to a total adequate order on tuples of words.

In the following we develop this approach. We start with a useful notational convention:

Notation 3. We write $[\mathbf{w}] = [\mathbf{w}_1, \dots, \mathbf{w}_n]$ to denote that $\mathbf{w}_1, \dots, \mathbf{w}_n$ are the projections of \mathbf{w} onto $\mathbf{T}_1, \dots, \mathbf{T}_n$ and that $[\mathbf{w}]$ is the unique trace with this property.

Example 4.47. For instance, we have

$$[\langle t_1, \epsilon \rangle \langle \epsilon, u_1 \rangle \langle t_3, u_2 \rangle \langle t_5, \epsilon \rangle \langle \epsilon, u_3 \rangle] = [\begin{array}{l} \langle t_1, \epsilon \rangle \langle t_3, u_2 \rangle \langle t_5, \epsilon \rangle , \\ \langle \epsilon, u_1 \rangle \langle t_3, u_2 \rangle \langle \epsilon, u_3 \rangle \end{array}] .$$

The following little proposition shows that the prefix order on words and the prefix order on traces fit nicely with each other.

Proposition 4.48. *If $[\mathbf{w}] = [\mathbf{w}_1, \dots, \mathbf{w}_n]$ and $[\mathbf{w}'] = [\mathbf{w}'_1, \dots, \mathbf{w}'_n]$ then $[\mathbf{w}][\mathbf{w}'] = [\mathbf{w}_1 \mathbf{w}'_1, \dots, \mathbf{w}_n \mathbf{w}'_n]$.*

Proof. By definition we have $[\mathbf{w}][\mathbf{w}'] = [\mathbf{w} \mathbf{w}']$. Since for every $i \in \{1, \dots, n\}$ the projection of $\mathbf{w} \mathbf{w}'$ onto \mathbf{T}_i is $\mathbf{w}_i \mathbf{w}'_i$, we are done. \square

We are now ready to define the “lifting” procedure and prove its correctness.

Definition 4.49. *Let \prec be a total search strategy on \mathbf{T}^* . The distributed strategy \prec^d is the total search strategy on $[\mathbf{T}^*]$ defined as follows. Given $[\mathbf{w}] = [\mathbf{w}_1, \dots, \mathbf{w}_n]$ and $[\mathbf{w}'] = [\mathbf{w}'_1, \dots, \mathbf{w}'_n]$, we have $[\mathbf{w}] \prec^d [\mathbf{w}']$ if there is an index $i \in \{1, \dots, n\}$ such that $\mathbf{w}_i \prec \mathbf{w}'_i$, and $\mathbf{w}_j = \mathbf{w}'_j$ for every $1 \leq j < i$.*

By Prop. 4.48, and since \prec refines the prefix order on words, \prec^d refines the prefix order on traces. Moreover, if \prec is adequate and total then so is \prec^d :

Theorem 4.50. *If \prec is a total adequate strategy on \mathbf{T}^* , then \prec^d is a total adequate strategy on $[\mathbf{T}^*]$.*

Proof. By the definition of an adequate strategy and Prop. 4.42, it suffices to prove that \prec^d is preserved by extensions. Let $[\mathbf{w}] = [\mathbf{w}_1, \dots, \mathbf{w}_n]$ and $[\mathbf{w}'] = [\mathbf{w}'_1, \dots, \mathbf{w}'_n]$ be traces such that $[\mathbf{w}] \prec^d [\mathbf{w}']$, and let $[\mathbf{w}''] = [\mathbf{w}''_1, \dots, \mathbf{w}''_n]$ be an arbitrary trace. By definition, there is an index i such that $\mathbf{w}_i \prec \mathbf{w}'_i$ and $\mathbf{w}_j = \mathbf{w}'_j$ for every $j < i$. Then we have $\mathbf{w}_j \mathbf{w}''_j = \mathbf{w}'_j \mathbf{w}''_j$ for every $j < i$ and, since \prec is preserved by extensions, $\mathbf{w}_i \mathbf{w}''_i \prec \mathbf{w}'_i \mathbf{w}''_i$. So $\mathbf{w} \mathbf{w}'' \prec \mathbf{w}' \mathbf{w}''$. \square

So to finish our quest for a total adequate strategy on $[\mathbf{T}^*]$ we just have to find a total adequate strategy on \mathbf{T}^* . But this is easy:

Definition 4.51. Let $<_a$ be the alphabetical order on \mathbf{T} . The lexicographic strategy on \mathbf{T}^* , denoted by \prec_l , is defined as follows: $\mathbf{w} \prec_l \mathbf{w}'$ if either \mathbf{w} is a proper prefix of \mathbf{w}' or there are words $\bar{\mathbf{w}}, \mathbf{v}, \mathbf{v}'$ and transitions \mathbf{a} and \mathbf{b} such that $\mathbf{a} <_a \mathbf{b}$, $\mathbf{w} = \bar{\mathbf{w}} \mathbf{a} \mathbf{v}$, and $\mathbf{w}' = \bar{\mathbf{w}} \mathbf{b} \mathbf{v}'$. The size-lexicographic strategy on \mathbf{T}^* , denoted by \prec_{sl} , is defined as follows: $\mathbf{w} \prec_{sl} \mathbf{w}'$ if either $\mathbf{w} \prec_s \mathbf{w}'$ or $\mathbf{w} =_s \mathbf{w}'$ and $\mathbf{w} \prec_l \mathbf{w}'$.

$$\begin{array}{c} \mathbf{w} = \bar{\mathbf{w}} \mathbf{a} \mathbf{v} \\ \prec_l \qquad \wedge \\ \mathbf{w}' = \bar{\mathbf{w}} \mathbf{b} \mathbf{v}'. \end{array}$$

Theorem 4.52. \prec_{sl} is a **total adequate** strategy on \mathbf{T}^* , and so \prec_{sl}^d is a total adequate strategy on $[\mathbf{T}^*]$.

Proof. By Thm. 4.50 it suffices to prove the first part. Clearly, \prec_{sl} is a well-founded total order on \mathbf{T}^* . We show that it is preserved by extensions. Let $\mathbf{w}, \mathbf{w}', \mathbf{w}''$ be words such that $\mathbf{w} \prec_{sl} \mathbf{w}'$. There are two possible cases:

- $\mathbf{w} \prec_s \mathbf{w}'$. Then $\mathbf{w} \mathbf{w}'' \prec_s \mathbf{w}' \mathbf{w}''$ and so $\mathbf{w} \mathbf{w}'' \prec_{sl} \mathbf{w}' \mathbf{w}''$.
- $\mathbf{w} =_s \mathbf{w}'$ and $\mathbf{w} \prec_l \mathbf{w}'$. Then there are words $\bar{\mathbf{w}}, \mathbf{v}, \mathbf{v}'$ and transitions \mathbf{a} and \mathbf{b} such that $\mathbf{a} <_a \mathbf{b}$, $\mathbf{w} = \bar{\mathbf{w}} \mathbf{a} \mathbf{v}$ and $\mathbf{w}' = \bar{\mathbf{w}} \mathbf{b} \mathbf{v}'$. But then we have $\mathbf{w} \mathbf{w}'' = \bar{\mathbf{w}} \mathbf{a} \mathbf{v} \mathbf{w}'' \prec_l \bar{\mathbf{w}} \mathbf{b} \mathbf{v}' \mathbf{w}'' = \mathbf{w}' \mathbf{w}''$, and so $\mathbf{w} \mathbf{w}'' \prec_{sl} \mathbf{w}' \mathbf{w}''$.

Since \prec_{sl}^d is a total adequate order on $[\mathbf{T}^*]$, it is also complete, and so the search procedure based on it will return the correct answer when applied to the product of Fig. 4.6 on p. 59.

Event	\mathbf{T}_1	\mathbf{T}_2	\mathbf{T}_3	\mathbf{T}_4
1	a	a	a	a
2	b	b	b	b
3	ac	ac	a	a
4	a	a	ad	ad
5	be	be	b	b
6	b	b	bf	bf
7	acg	ac	adg	ad
8	beg	be	bf	bf
9	be	beh	bf	bfh
10	ac	ach	ad	adh
11	acgi	achi	adgi	adhi
12	begi	behi	bf	bfhi

$\mathbf{T} = \{ \mathbf{a} = \langle a_1, a_2, a_3, a_4 \rangle, \mathbf{b} = \langle b_1, b_2, b_3, b_4 \rangle, \mathbf{c} = \langle c_1, c_2, \epsilon, \epsilon \rangle,$
 $\mathbf{d} = \langle \epsilon, \epsilon, d_3, d_4 \rangle, \mathbf{e} = \langle e_1, e_2, \epsilon, \epsilon \rangle, \mathbf{f} = \langle \epsilon, \epsilon, f_3, f_4 \rangle,$
 $\mathbf{g} = \langle g_1, \epsilon, g_3, \epsilon \rangle, \mathbf{h} = \langle \epsilon, h_2, \epsilon, h_4 \rangle, \mathbf{i} = \langle i_1, i_2, i_3, i_4 \rangle \}$

$\mathbf{G} = \{ \mathbf{i} \}$

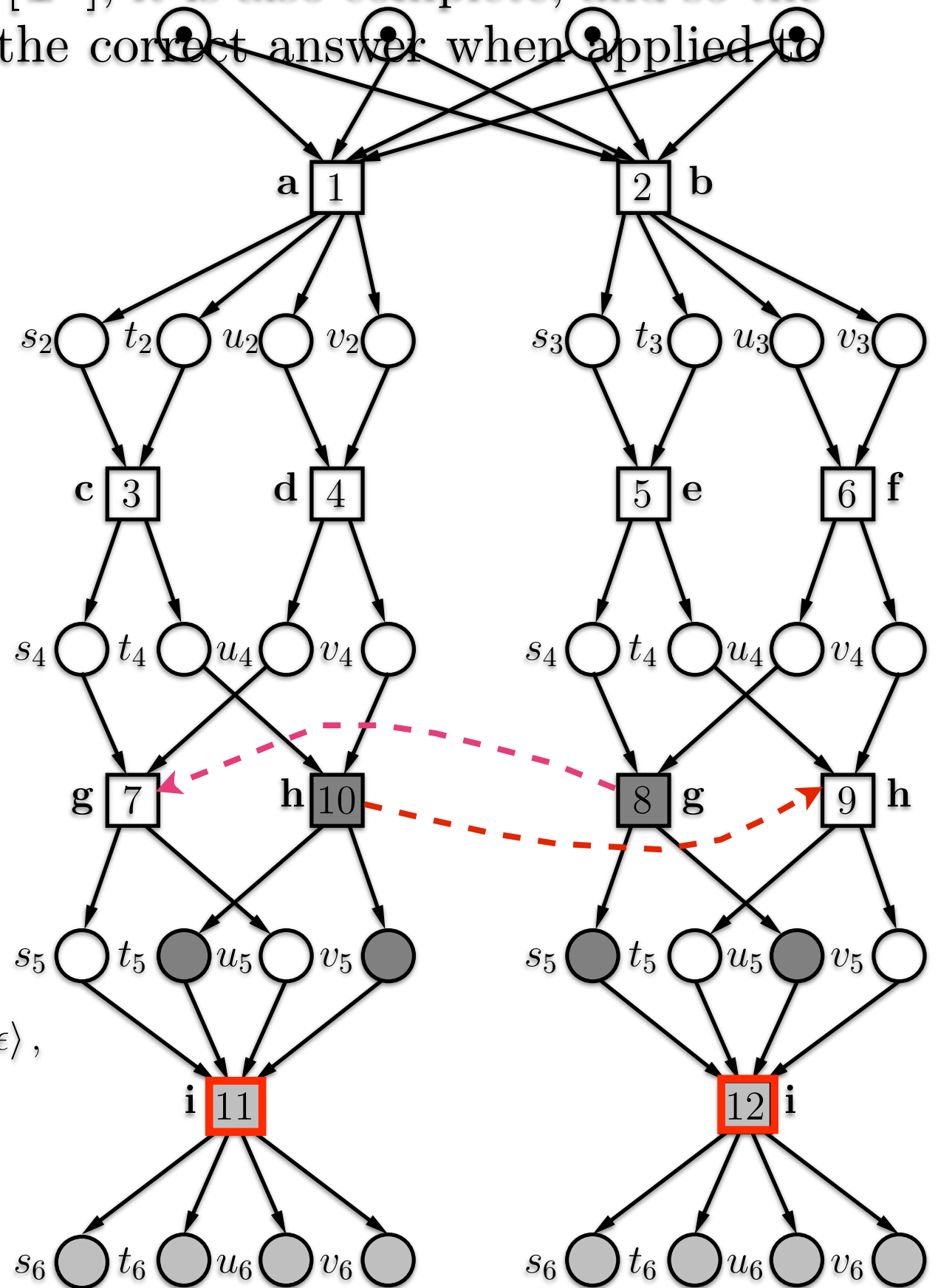
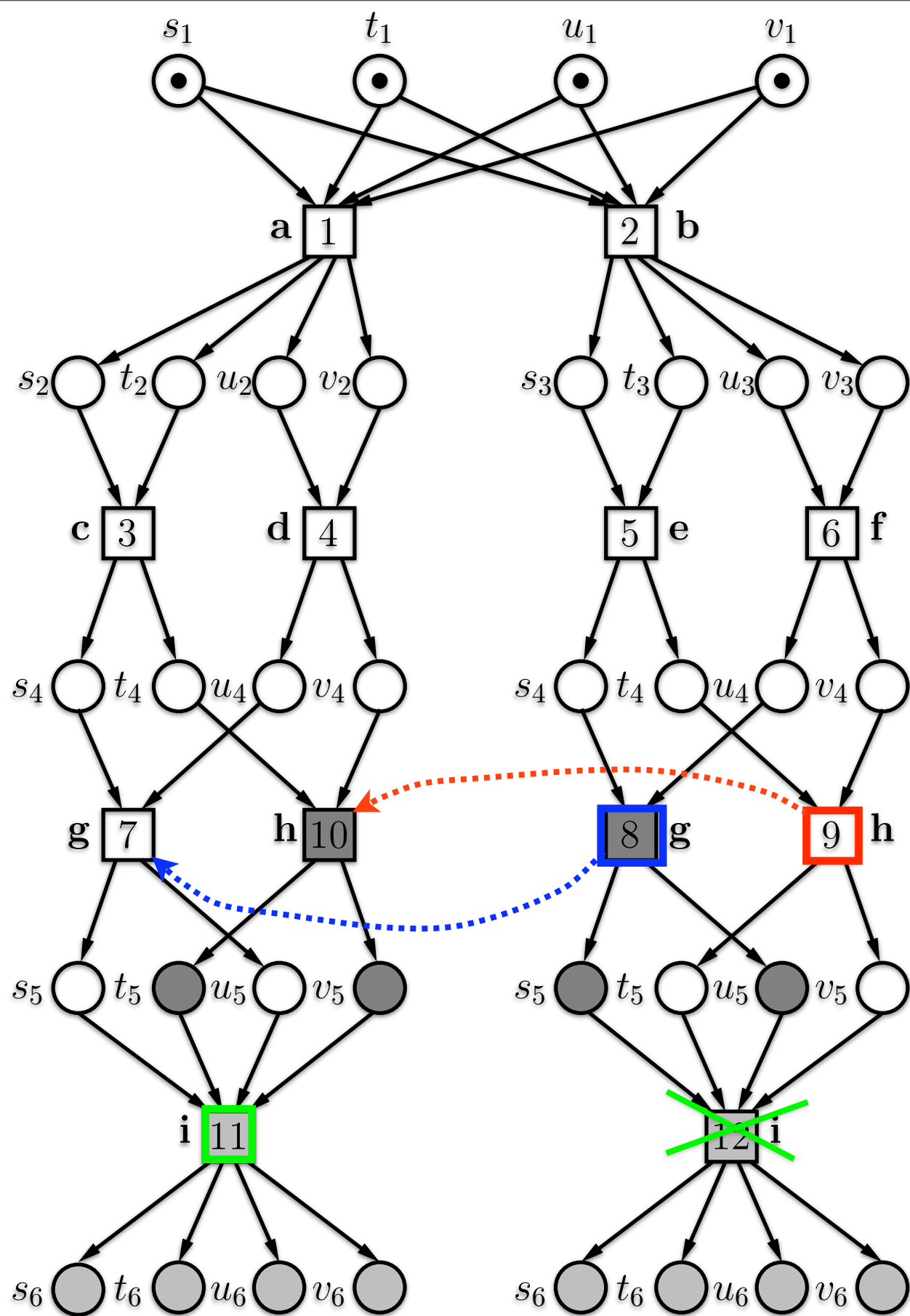


Fig. 4.8. Unfolding of the product of Fig. 4.6

Event	T ₁	T ₂	T ₃	T ₄
1	a	a	a	a
2	b	b	b	b
3	ac	ac	a	a
4	a	a	ad	ad
5	be	be	b	b
6	b	b	bf	bf
7	acg	ac	adg	ad
8	beg	be	bf	bf
9	be	beh	bf	bfh
10	ac	ach	ad	adh
11	acgi	achi	adgi	adhi
12	begi	behi	bfgi	bfhi

Event	T ₁	T ₂	T ₃	T ₄
1	a	a	a	a
4	a	a	ad	ad
2	b	b	b	b
6	b	b	bf	bf
3	ac	ac	a	a
10	ac	ach	ad	adh
5	be	be	b	b
9	be	beh	bf	bfh
7	acg	ac	adg	ad
8	beg	be	bf	bf
11	acgi	achi	adgi	adhi
12	begi	behi	bfgi	bfhi



Event	T_1	T_2	T_3	T_4
1	a	a	a	a
4	a	a	ad	ad
2	b	b	b	b
6	b	b	bf	bf
3	ac	ac	a	a
10	ac	ach	ad	adh
5	be	be	b	b
9	be	beh	bf	bfh
7	acg	ac	adg	ad
8	beg	be	bfg	bf
11	acgi	achi	adgi	adhi
12	begi	behi	bfgi	bfhi

Fig. 4.8. Unfolding of the product of Fig. 4.6

More on the Executability Problem

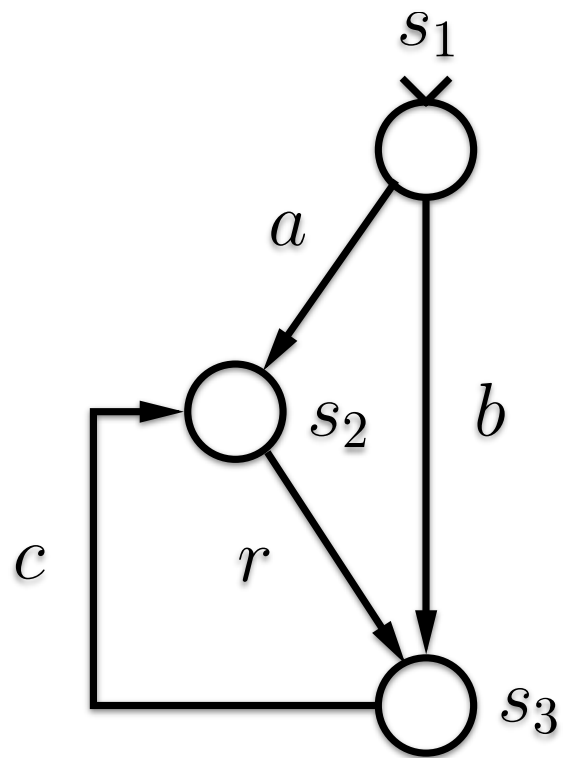
In this chapter we present some additional results on the executability problem, and in particular on adequate strategies, that are not used in the rest of the book. The reader interested in the essentials of the model checking procedure for LTL can safely skip them and move to Chap. 6.

In Sect. 5.1 we introduce *complete prefixes*. These are prefixes of the unfolding which, loosely speaking, contain all reachable states. Once a complete prefix is computed, it can be used to solve many different verification problems.

The other sections of the chapter present further results on adequate strategies. Section 5.2 introduces a second total adequate strategy, different from the distributed version of the size-lexicographic strategy that was defined in the previous chapter. Section 5.3 generalizes the concept of breadth-first and depth-first search to products. Finally, Sect. 5.4 proves Prop. 4.42, stating that every strategy on Mazurkiewicz traces preserved by extensions is well-founded.

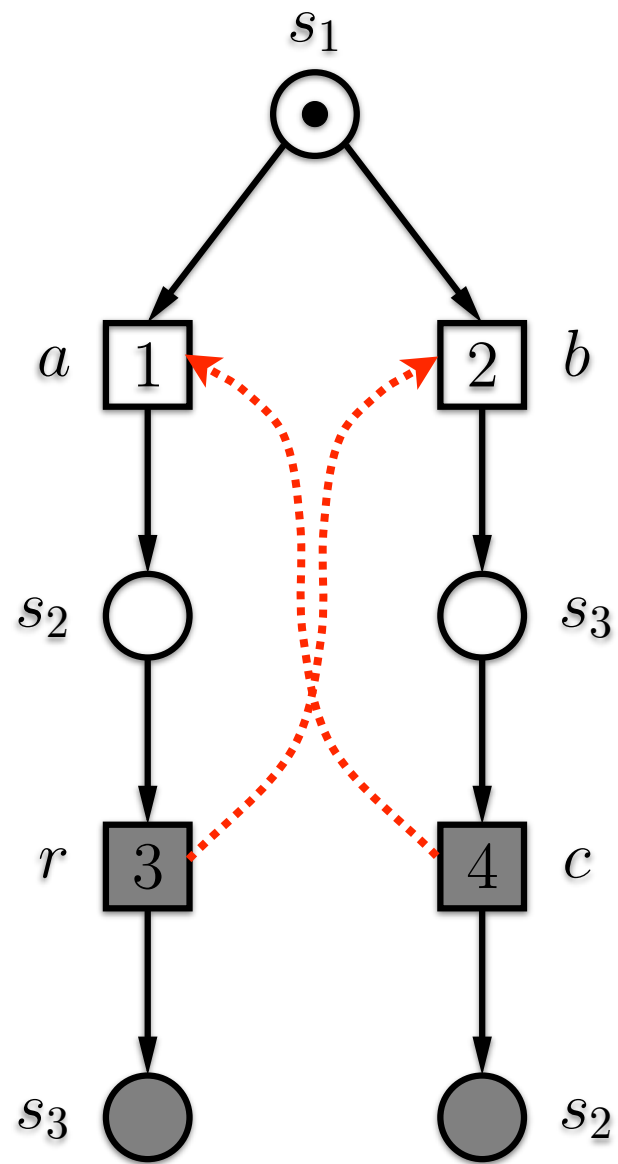
Search Procedures for the Repeated Executability Problem

- **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} ?



$$R = \{r\}$$

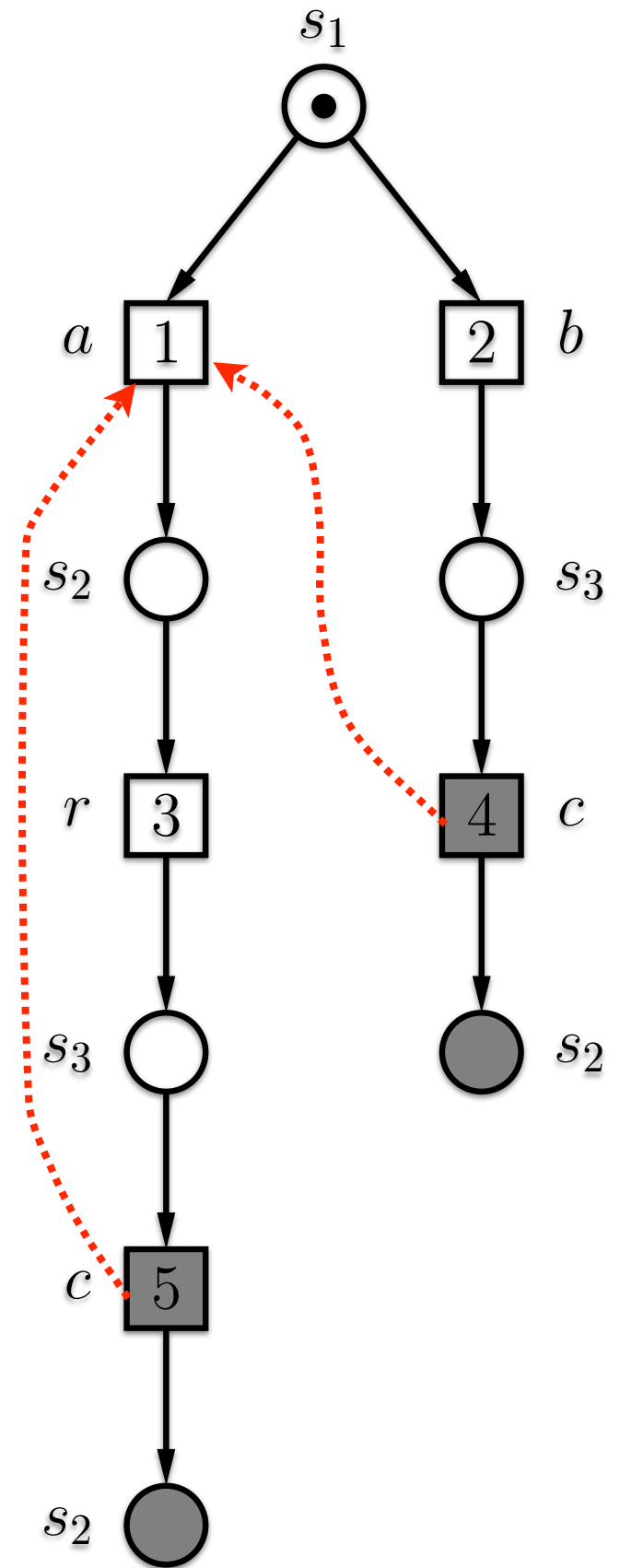
(a)



(c)

breadth-first strategy

$$\#_R(e') \geq \#_R(e)$$



Definition 6.1. Let \prec be a search strategy on T^* . An event e is feasible if no event $e' < e$ is a terminal. A feasible event e is a terminal if there exists a feasible event $e' \prec e$, called the companion of e , such that $St(e') = St(e)$ and at least one of (a) $e' < e$ or (b) $\#_R(e') \geq \#_R(e)$ holds, where $\#_R(e)$ denotes the number of occurrences of transitions of R in the history $H(e)$.

A terminal is successful if it satisfies (a) and $\#_R(e') < \#_R(e)$ holds. The \prec -final prefix is the prefix of the unfolding of \mathcal{A} containing the feasible events.

Proposition 6.3. The search scheme of Def. 6.1 is well-defined for every strategy \prec . Moreover, the \prec -final prefix is finite.

Theorem 6.4. The search scheme of Def. 6.1 is complete for every strategy.

As for the executability problem, one way to reduce the size of the final prefix is to require the strategy \prec to be a total order. However, in this case the number of non-terminals of the final prefix may grow quadratically in the number of states of \mathcal{A} .

Theorem 6.5. *If \prec is a total order on T^* , then the \prec -final prefix of Def. 6.1 has at most $|S|^2$ non-terminal events.*

6.2 Search Scheme for Products

We fix a product $\mathbf{A} = \langle \mathcal{A}_1, \dots, \mathcal{A}_n, \mathbf{T} \rangle$, where $\mathcal{A}_i = \langle S_i, T_i, \alpha_i, \beta_i, is_i \rangle$, and a set of global transitions $\mathbf{R} \subseteq \mathbf{T}$. The problem to solve is whether some infinite global history of \mathbf{A} executes transitions of \mathbf{R} infinitely often. We call the elements of \mathbf{R} (global) \mathbf{R} -transitions, and call the events labeled by them \mathbf{R} -events.

We generalize the search scheme of Def. 6.1 to products. Following the ideas introduced in Sect. 4.4, we replace $St(e)$ by $\mathbf{St}(e)$, and $H(e)$ by $\mathbf{H}(e)$. The remaining question is how to generalize $\#_R(e)$. Recall that in the transition system case we defined $\#_R(e)$ as the number of occurrences of transitions of R in $H(e)$.

Definition 6.6. *Let e be an event, and let \mathbf{h} be any history of the trace $\mathbf{H}(e)$. We denote by $\#\mathbf{R}([\mathbf{h}])$ the number of occurrences of \mathbf{R} -transitions in \mathbf{h} , and define $\#\mathbf{R}(e) = \#\mathbf{R}([\mathbf{h}])$.*

Notice that $\#\mathbf{R}(e)$ is well-defined because every transition occurs the same number of times in all histories of $\mathbf{H}(e)$.

We are now ready to define the search scheme for products.

Definition 6.8. Let \prec be a search strategy on $[\mathbf{T}^*]$. An event e is feasible if no event $e' < e$ is a terminal. A feasible event e is a terminal if there exists a feasible event $e' \prec e$, called the companion of e , such that $\mathbf{St}(e') = \mathbf{St}(e)$ and at least one of (a) $e' < e$ or (b) $\#_{\mathbf{R}}(e') \geq \#_{\mathbf{R}}(e)$ holds.

A terminal is successful if it satisfies (a) and $\#_{\mathbf{R}}(e') < \#_{\mathbf{R}}(e)$. The \prec -final prefix is the prefix of the unfolding of \mathbf{A} containing the feasible events.

Theorem 6.11. The search scheme of Def. 6.8 is complete for every adequate strategy.

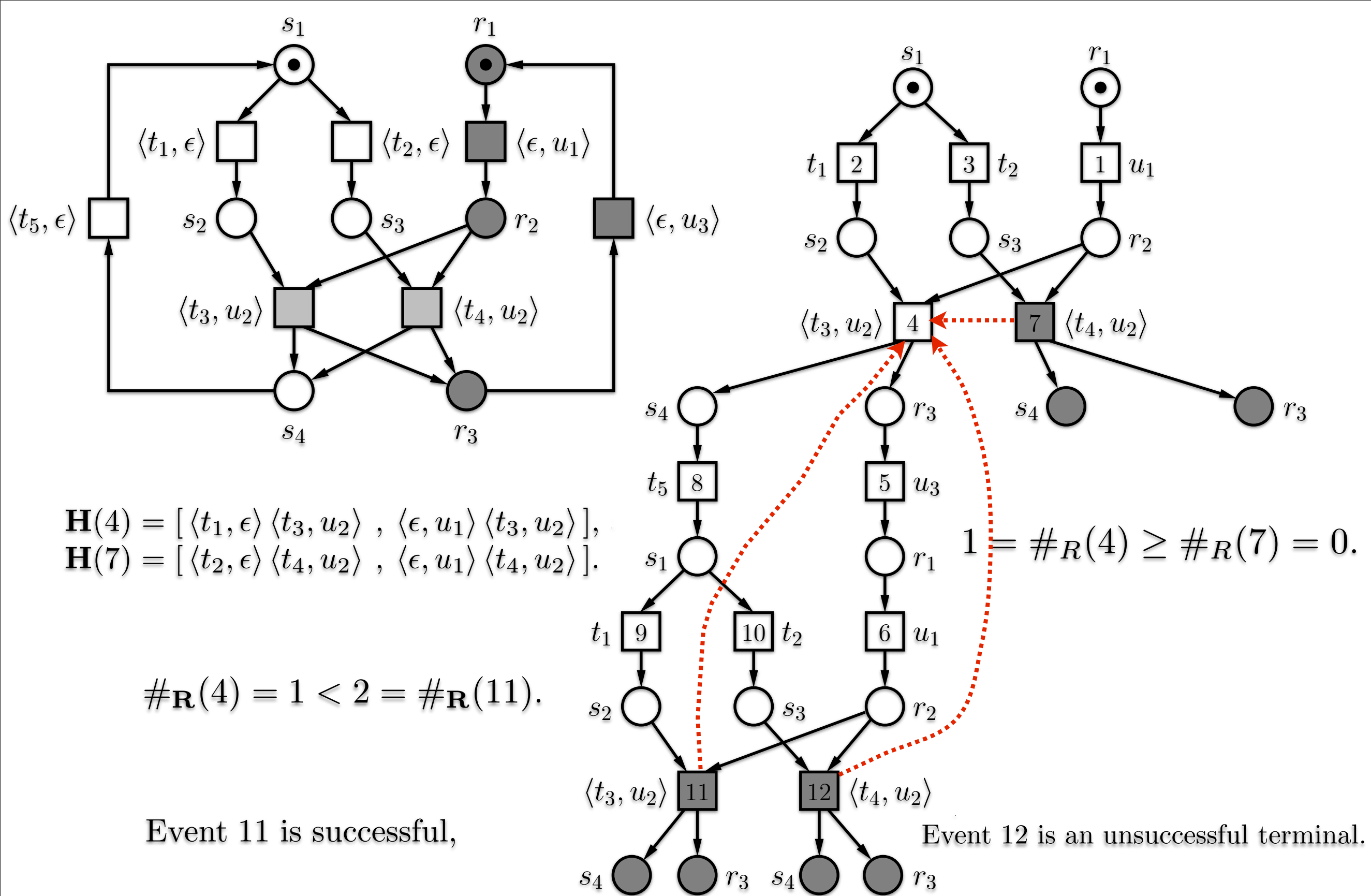


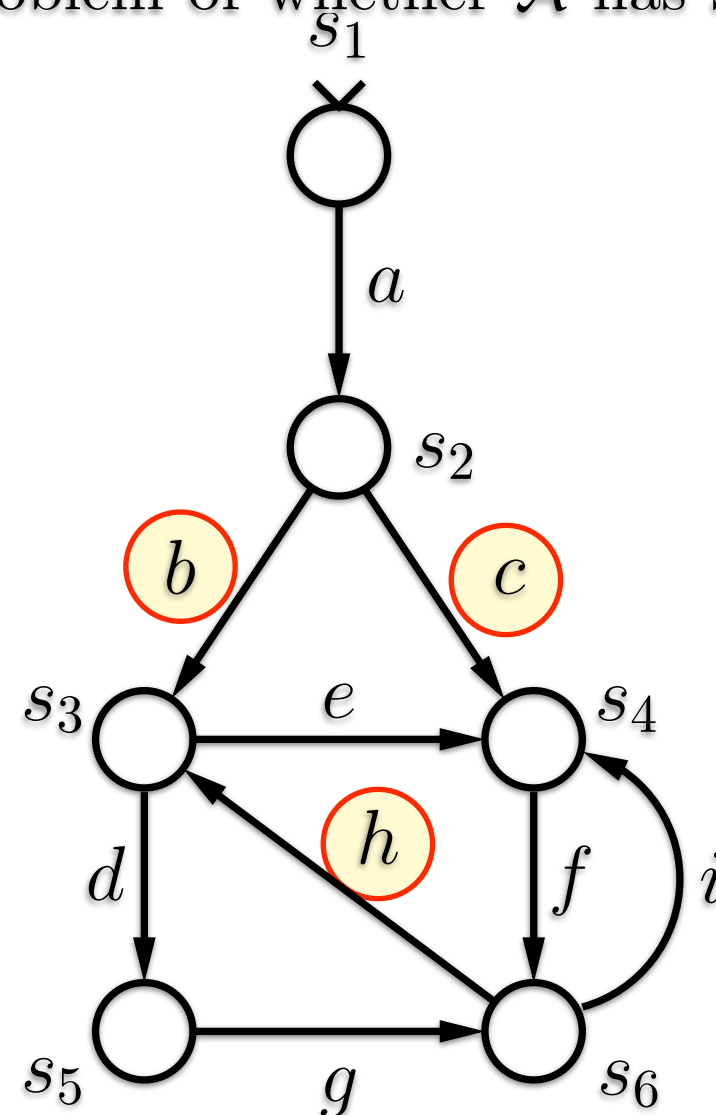
Fig. 6.2. Repeated executability final prefix of the product of Fig. 2.2 on p. 7 for $\mathbf{R} = \{ \langle t_1, \epsilon \rangle \}$

distributed size-lexicographic strategy, i.e., $\prec = \prec_{sl}^d$. 57

Search Procedures for the Livelock Problem

We fix a transition system $\mathcal{A} = \langle S, T, \alpha, \beta, is \rangle$, and partition the set T into a set V of *visible* and a set $I = T \setminus V$ of *invisible* transitions. We also fix a set $L \subseteq V$ of *livelock monitors*.

A *livelock* is an infinite history of the form $h t c$, where $h \in T^*$ is a finite history, $t \in L$ is a livelock monitor, and $c \in I^\omega$ is an infinite computation containing only invisible transitions. We call the occurrence of t after the history h the *livelock's root*. Intuitively, livelocks are undesirable behaviors in which right after the livelock's root the system enters an infinite loop of unobservable actions. We wish to solve the problem of whether \mathcal{A} has some livelock.



$$V = \{b, c, h\}$$

$$L = \{b, c\}$$

Model Checking LTL

The set of formulas of Linear Temporal Logic (LTL) over a given nonempty set AP of *atomic propositions* is inductively defined as follows:

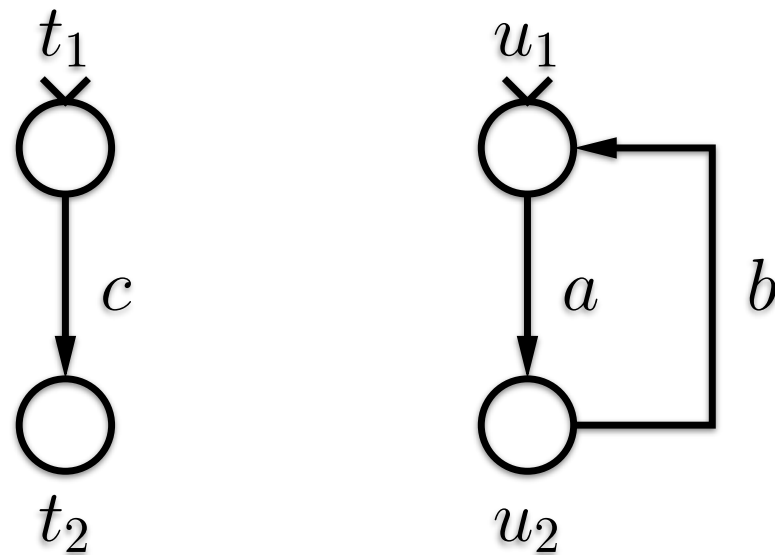
- every atomic proposition is an LTL formula,
- if ψ_1 is an LTL formula, then $\neg\psi_1$ and $\mathbf{X}\psi_1$ are LTL formulas; and
- if ψ_1, ψ_2 are LTL formulas, then $\psi_1 \vee \psi_2$ and $\psi_1 \mathbf{U} \psi_2$ are LTL formulas.

We employ the usual shorthands for LTL formulas: $true = \neg p \vee p$ for an arbitrary $p \in AP$, $false = \neg true$, $\psi_1 \wedge \psi_2 = \neg(\neg\psi_1 \vee \neg\psi_2)$, $\psi_1 \Rightarrow \psi_2 = \neg\psi_1 \vee \psi_2$, $\psi_1 \mathbf{R} \psi_2 = \neg(\neg\psi_1 \mathbf{U} \neg\psi_2)$, $\mathbf{F} \psi_1 = true \mathbf{U} \psi_1$, and $\mathbf{G} \psi_1 = false \mathbf{R} \psi_1$.

$$\begin{aligned} (\{p\} \emptyset)^\omega &\models \mathbf{G} (p \Rightarrow \mathbf{X} \neg p) \\ (\{p\} \emptyset)^\omega &\not\models \mathbf{F} (p \wedge \mathbf{X} p) \end{aligned}$$

$$\begin{aligned} \emptyset \emptyset \{p\} \{p\} (\emptyset)^\omega &\not\models \mathbf{G} (p \Rightarrow \mathbf{X} \neg p) \\ \emptyset \emptyset \{p\} \{p\} (\emptyset)^\omega &\models \mathbf{F} (p \wedge \mathbf{X} p) \end{aligned}$$

Example 8.2. Consider the product of Fig. 8.1. We consider LTL over the set of atomic propositions $AP = \{t_1, t_2, u_1, u_2\}$. The sequence $\mathbf{h} = \mathbf{a b c}(\mathbf{a b})^\omega$ is an infinite history. The sequence of global states visited along its execution is $\langle t_1, u_1 \rangle \langle t_1, u_2 \rangle \langle t_1, u_1 \rangle (\langle t_2, u_1 \rangle \langle t_2, u_2 \rangle)^\omega$, and we have $\pi(\mathbf{h}) = \{t_1, u_1\} \{t_1, u_2\} \{t_1, u_1\} (\{t_2, u_1\} \{t_2, u_2\})^\omega$.



$$\mathbf{T} = \{\mathbf{a} = \langle \epsilon, a \rangle, \mathbf{b} = \langle \epsilon, b \rangle, \mathbf{c} = \langle c, \epsilon \rangle\}$$

Fig. 8.1. A running example for LTL model checking

$$\pi(\mathbf{h}) \not\models \mathbf{G} (u_1 \Rightarrow \mathbf{X} \neg u_1) \quad \text{and} \quad \pi(\mathbf{h}) \models \mathbf{F} (u_1 \wedge \mathbf{X} u_1).$$

So, by definition, we have

$$\mathbf{h} \not\models \mathbf{G} (u_1 \Rightarrow \mathbf{X} \neg u_1) \quad \text{and} \quad \mathbf{h} \models \mathbf{F} (u_1 \wedge \mathbf{X} u_1).$$

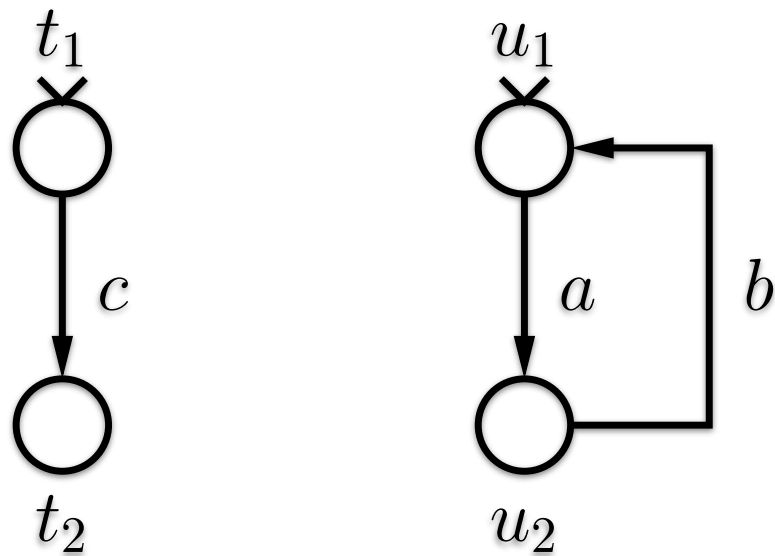
So far we know how to interpret a formula ψ on the infinite histories of the product \mathbf{A} , but it is convenient to extend the interpretation to what we call the ψ -histories of \mathbf{A} . The reason is that in order to solve the model checking problem we will later construct an automaton accepting exactly these histories.

Let AP_ψ be the set of atomic propositions that appear in ψ . A ψ -state is a tuple $\mathbf{r} = \langle r_1, \dots, r_n \rangle$ such that for every $i \in \{1, \dots, n\}$ either $r_i \in S_i \cap AP_\psi$ or $s_i = \perp$, where \perp is a special symbol. Given a global state $\mathbf{s} = \langle s_1, \dots, s_n \rangle$, we assign to it a ψ -state $\mathbf{s}_\psi = \langle s_{1\psi}, \dots, s_{n\psi} \rangle$ as follows. For every $i \in \{1, \dots, n\}$,

$$s_{i\psi} = \begin{cases} s_i & \text{if } s_i \in AP_\psi, \text{ and} \\ \perp & \text{otherwise.} \end{cases}$$

So, intuitively, \mathbf{s}_ψ is the information on the global state \mathbf{s} available to an observer that can only see the local states of AP_ψ .

A tuple $\langle \mathbf{r}, \mathbf{t}, \mathbf{r}' \rangle$, where \mathbf{r}, \mathbf{r}' are ψ -states and \mathbf{t} is a global transition, is a ψ -step if there exists a step $\langle \mathbf{s}, \mathbf{t}, \mathbf{s}' \rangle$ such that $\mathbf{r} = \mathbf{s}_\psi$ and $\mathbf{r}' = \mathbf{s}'_\psi$. We define ψ -computations and ψ -histories by taking the definitions of computation and history, respectively, and replacing steps by ψ -steps. A sequence $\mathbf{t}_1 \dots \mathbf{t}_k$ of global transitions is a ψ -computation if there is a sequence $\mathbf{r}_0, \dots, \mathbf{r}_k$ of ψ -states such that $\langle \mathbf{r}_{i-1}, \mathbf{t}_i, \mathbf{r}_i \rangle$ is a ψ -step for every $i \in \{1, \dots, k\}$. A ψ -computation is a ψ -history if one can choose the sequence $\mathbf{r}_0, \dots, \mathbf{r}_k$ such that $\mathbf{r}_0 = \mathbf{is}_\psi$. Infinite ψ -computations and infinite ψ -histories are defined analogously. To gain some intuition, imagine that an observer can only see the local states of AP_ψ and the transitions having them as source or target states. A sequence of transitions is a ψ -history if this observer cannot conclude that it is not a history by just observing the marking changes in all the places in AP_ψ .



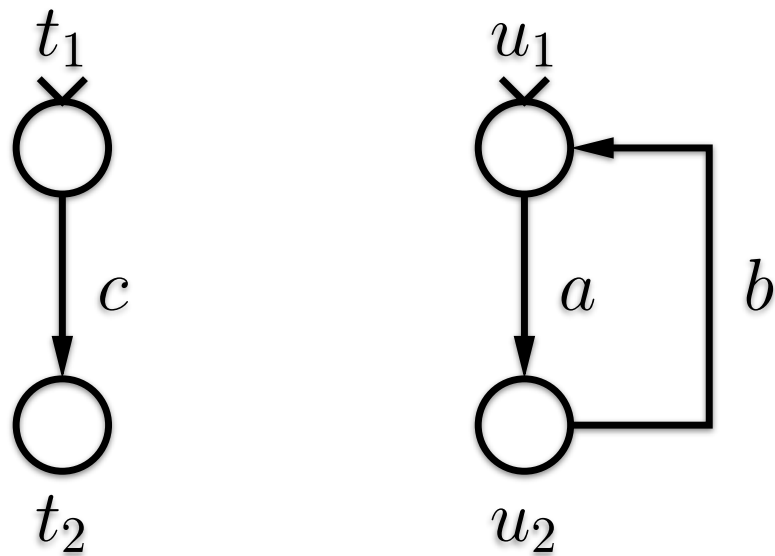
$$\mathbf{T} = \{\mathbf{a} = \langle \epsilon, a \rangle, \mathbf{b} = \langle \epsilon, b \rangle, \mathbf{c} = \langle c, \epsilon \rangle\}$$

Fig. 8.1. A running example for LTL model checking

Example 8.4. Consider again the product of Fig. 8.1, and assume that we have $AP_\psi = \{u_1\}$ (the exact formula ψ is irrelevant for this example). The triple $\langle \langle t_1, u_1 \rangle, \mathbf{c}, \langle t_2, u_1 \rangle \rangle$ is a step, and $\langle \langle \perp, u_1 \rangle, \mathbf{c}, \langle \perp, u_1 \rangle \rangle$ is its corresponding ψ -step. It follows that $\mathbf{c c}$ is a ψ -history because of the two ψ -steps

$$\langle \langle \perp, u_1 \rangle, \mathbf{c}, \langle \perp, u_1 \rangle \rangle \langle \langle \perp, u_1 \rangle, \mathbf{c}, \langle \perp, u_1 \rangle \rangle .$$

Observe however that $\mathbf{c c}$ is not a history.



$$\mathbf{T} = \{\mathbf{a} = \langle \epsilon, a \rangle, \mathbf{b} = \langle \epsilon, b \rangle, \mathbf{c} = \langle c, \epsilon \rangle\}$$

Fig. 8.1. A running example for LTL model checking

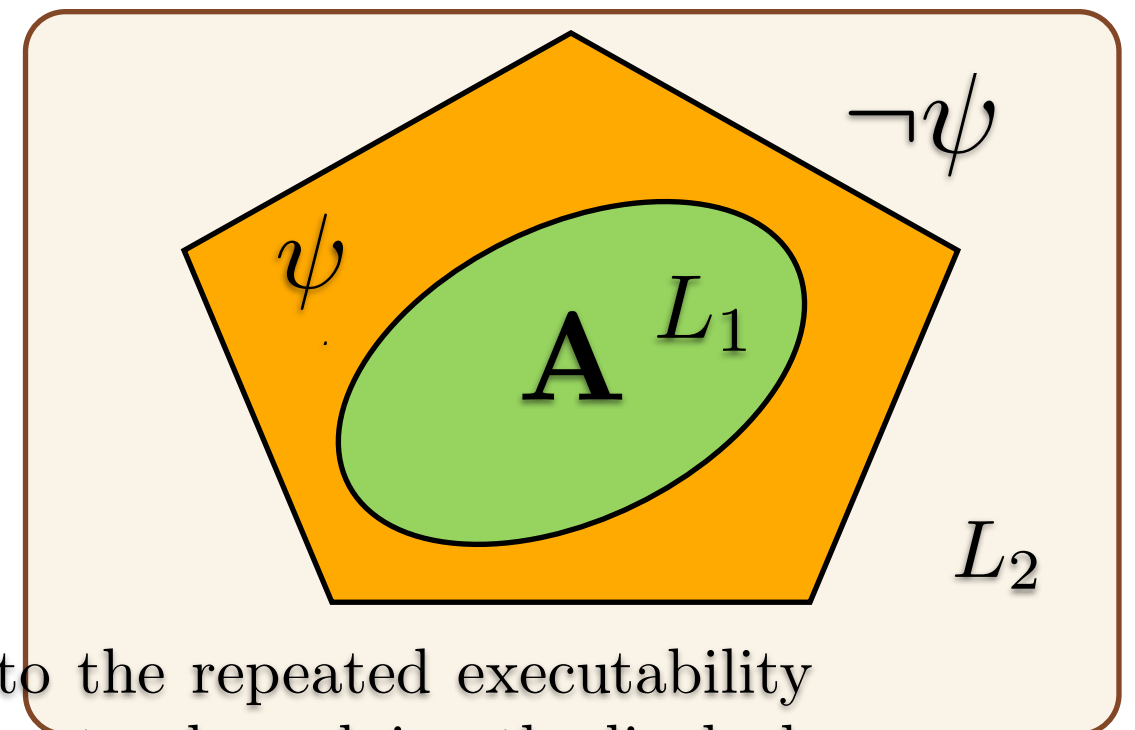
On the other hand, the sequence $\mathbf{a a}$ is not a ψ -history. To see why, assume there are ψ -steps $\langle \mathbf{r}_0, \mathbf{a}, \mathbf{r}_1 \rangle$ and $\langle \mathbf{r}_1, \mathbf{a}, \mathbf{r}_2 \rangle$ such that \mathbf{r}_0 is the initial ψ -state, i.e., $\mathbf{r}_0 = \langle \perp, u_1 \rangle$. By the definition of ψ -step we have $\mathbf{r}_1 = \langle \perp, \perp \rangle$. But, since \mathbf{a} can only occur at global states such that the second component is in state u_1 , there is no \mathbf{r}_2 such that $\langle \mathbf{r}_1, \mathbf{a}, \mathbf{r}_2 \rangle$ is a ψ -step.

As in the case of histories, it is easy to see that given an infinite ψ -history $\sigma = \mathbf{t}_1 \mathbf{t}_2 \mathbf{t}_3 \dots$ of \mathbf{A} , there is a unique sequence $\mathbf{r}_0 \mathbf{r}_1 \mathbf{r}_2 \dots$ of ψ -states such that $\mathbf{r}_0 = \mathbf{is}_\psi$ and $\langle \mathbf{r}_{i-1}, \mathbf{t}_i, \mathbf{r}_i \rangle$ is a ψ -step of \mathbf{A} for every $i \geq 1$. We denote this sequence by $\pi_\psi(\sigma)$, and call it *the ψ -sequence of σ* . We say that σ satisfies ψ , denoted by $\sigma \models \psi$, if $\pi_\psi(\sigma) \models \psi$.

8.3 Testers for LTL Properties

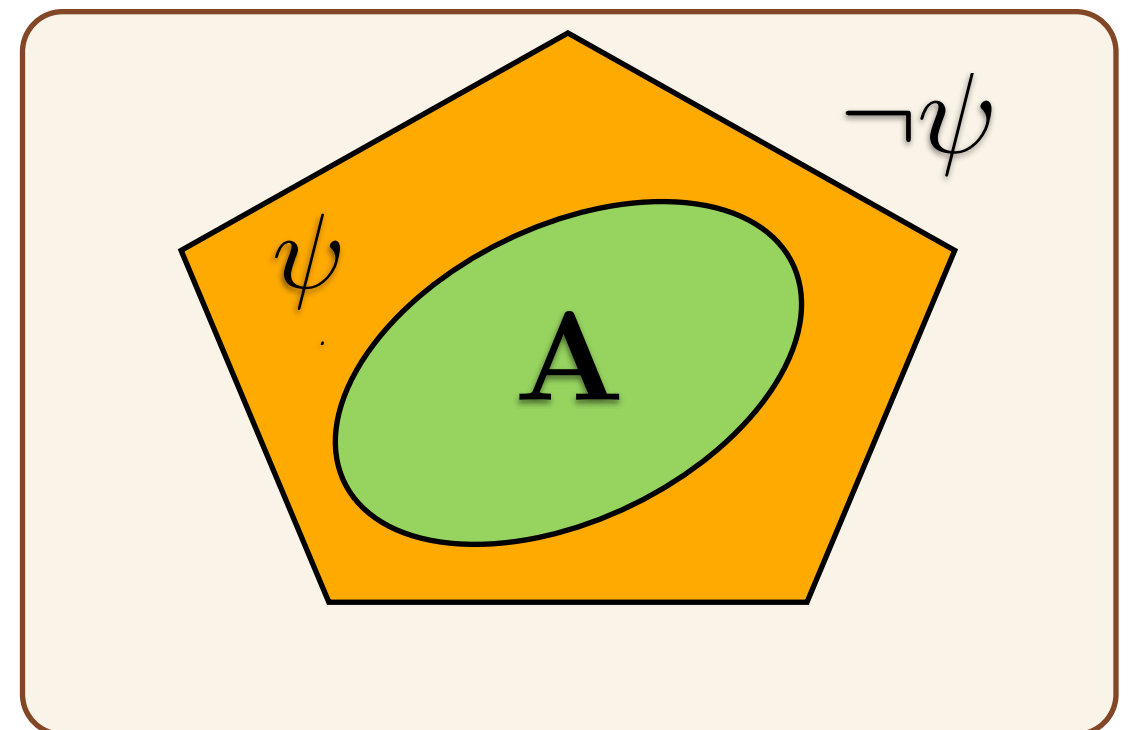
Deciding whether *all* infinite histories of a product \mathbf{A} satisfy ψ is equivalent to deciding whether *some* infinite history violates ψ , which in turn is equivalent to deciding if some infinite history satisfies $\neg\psi$. The tester approach to the model checking problem reduces this last question to the simpler one of checking if some history of a new product, which depends on ψ , satisfies a *fixed* property.

- Construct a *tester* recognizing the set L_2 of all ψ -histories satisfying $\neg\psi$.
- Using this tester, construct a new product recognizing the intersection $L_1 \cap L_2$, i.e., the set of ψ -histories of \mathbf{A} violating ψ .
- Check whether $L_1 \cap L_2$ is empty or not.



We will see that the check can be reduced to the repeated executability problem. Then, in Sect. 8.5 we additionally resort to also solving the livelock problem for efficiency reasons. We know how both of these problems can be solved from the previous chapters.

The testers suitable for checking LTL properties are called Büchi testers. A *Büchi tester* of \mathbf{A} , or simply a *tester*, is a triple $\mathcal{BT} = (\mathcal{B}, F, \lambda)$, where $\mathcal{B} = (S, T, \alpha, \beta, is)$ is a transition system, $F \subseteq S$ is a set of *accepting states*, and $\lambda: T \rightarrow \mathbf{T}$ is a *labeling function* that assigns to each transition of \mathcal{B} a global transition of \mathbf{A} . A tester \mathcal{BT} *recognizes* an infinite sequence $\mathbf{t}_1 \mathbf{t}_2 \mathbf{t}_3 \dots \in \mathbf{T}^\omega$ if there is an infinite history $h = u_1 u_2 u_3 \dots$ of \mathcal{B} and an accepting state $s \in F$ such that $\mathbf{t}_i = \lambda(u_i)$ for every $i \geq 1$ and h visits the state s infinitely often, i.e., the sequence $\pi(h)$ contains infinitely many occurrences of s . The *language* of \mathcal{BT} is the set of words of \mathbf{T}^ω recognized by \mathcal{BT} .



Example 8.5. Figure 8.2 shows a tester of the product of Fig. 8.1 for the property $\mathbf{F} t_2$. The names of the tester transitions have been omitted; we show only the global transitions of the product they are labeled with. The states drawn using two circles are the accepting states, and states drawn with a single circle are the non-accepting states. The tester recognizes the sequence $\mathbf{c}(\mathbf{a} \mathbf{b})^\omega$, but not $(\mathbf{a} \mathbf{b})^\omega$.

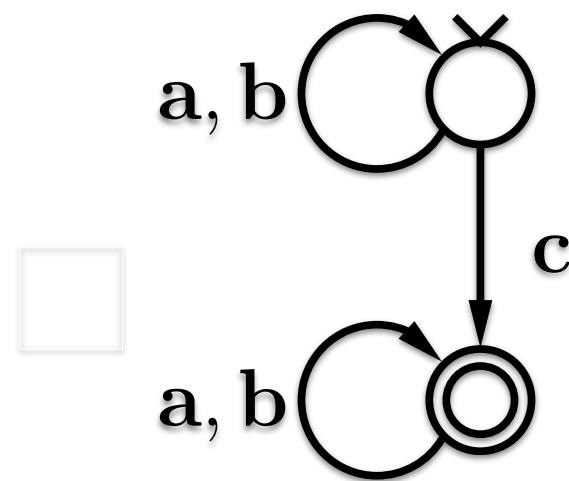
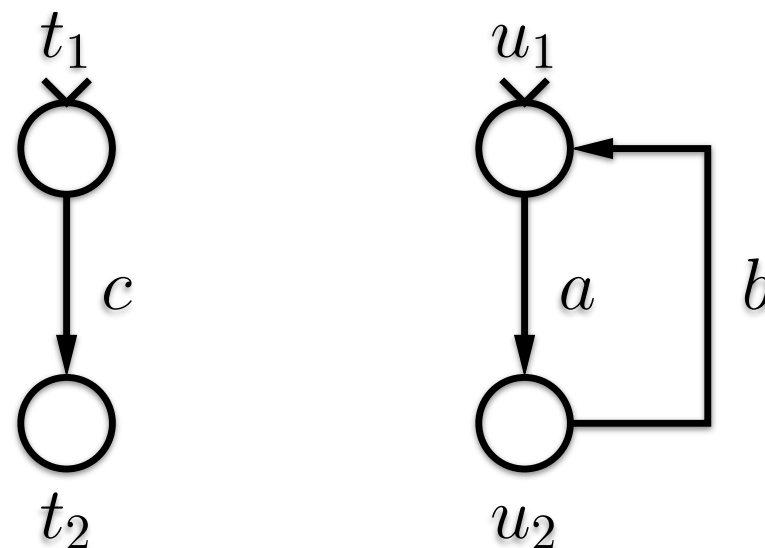


Fig. 8.2. A tester of the product of Fig. 8.1 for $\mathbf{F} t_2$



$$\mathbf{T} = \{\mathbf{a} = \langle \epsilon, a \rangle, \mathbf{b} = \langle \epsilon, b \rangle, \mathbf{c} = \langle c, \epsilon \rangle\}$$

Let ψ be a formula of LTL. A tester \mathcal{BT} tests the property ψ or is a tester for ψ if it recognizes the infinite ψ -histories of \mathbf{A} that satisfy ψ .

In the rest of this chapter we show that every LTL formula over AP has a Büchi tester. This is a very well-known topic in the area of model checking, and there exist many different constructions. We sketch a very simple one, without giving a formal proof of correctness. Readers familiar with LTL to Büchi automata translations may wish to jump directly to Sect. 8.4.

8.4 Model Checking with Testers: A First Attempt

We define a synchronization of a tester and product in which the tester observes all global transitions of the product. We show that this synchronization reduces the model checking problem to the repeated executability problem.

It is convenient to introduce some notation. Given a global transition $\mathbf{t} = \langle t_1, \dots, t_n \rangle$ of \mathbf{A} and a transition u of a tester \mathcal{BT} , we use $\langle \mathbf{t}, u \rangle$ as an abbreviation of the tuple $\langle t_1, \dots, t_n, u \rangle$. Similarly, given a global state $\mathbf{s} = \langle s_1, \dots, s_n \rangle$ of \mathbf{A} and a state r of \mathcal{BT} , we abbreviate $\langle s_1, \dots, s_n, r \rangle$ to $\langle \mathbf{s}, r \rangle$.

Definition 8.16. *Let $\mathcal{BT} = (\mathcal{B}, F, \lambda)$ be a tester. The full synchronization of \mathbf{A} and \mathcal{BT} is the product $\mathbf{A} \parallel \mathcal{BT} = \langle \mathcal{A}_1, \dots, \mathcal{A}_n, \mathcal{B}, \mathbf{U} \rangle$, where the synchronization constraint \mathbf{U} contains a global transition $\langle \mathbf{t}, u \rangle$ for every global transition \mathbf{t} of \mathbf{A} and every transition u of \mathcal{BT} such that $\lambda(u) = \mathbf{t}$.*

Given a sequence $\sigma = \langle \mathbf{t}_1, u_1 \rangle \langle \mathbf{t}_2, u_2 \rangle \langle \mathbf{t}_3, u_3 \rangle \dots$ of transitions of $\mathbf{A} \parallel \mathcal{BT}$, we call $\sigma_{\mathcal{A}} = \mathbf{t}_1 \mathbf{t}_2 \mathbf{t}_3 \dots$ the *projection of σ on \mathbf{A}* , and $\sigma_{\mathcal{BT}} = u_1 u_2 u_3 \dots$ the *projection of σ on \mathcal{BT}* . It follows immediately from the definition of full synchronization that σ is a history of $\mathbf{A} \parallel \mathcal{BT}$ if and only if $\sigma_{\mathcal{A}}$ and $\sigma_{\mathcal{BT}}$ are histories of \mathbf{A} and \mathcal{BT} , respectively. We say that σ *visits accepting states infinitely often* if there are infinitely many indices $i \geq 1$ such that the target state of the transition u_i is an accepting state of \mathcal{BT} .

Proposition 8.17. *Let $\mathcal{BT}_{\neg\psi}$ be a tester for $\neg\psi$. A history of \mathbf{A} violates ψ if and only if it is the projection on \mathbf{A} of an infinite history of $\mathbf{A} \parallel \mathcal{BT}_{\neg\psi}$ that visits accepting states infinitely often.*

Example 8.18. Consider the transition system \mathcal{A} shown in Fig. 8.4. Suppose we want to check whether \mathcal{A} satisfies $\psi_1 = \mathbf{F} (u_1 \wedge \mathbf{X} u_1)$, i.e., whether all global histories of \mathcal{A} eventually contain two consecutive time steps in which \mathcal{A} is in state u_1 . A Büchi tester $\mathcal{BT}_{\neg\psi_1} = \mathcal{BT}_{\mathbf{G} (u_1 \Rightarrow \mathbf{X} \neg u_1)}$ is shown in Fig. 8.5.

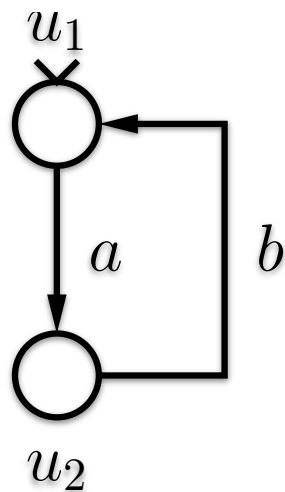


Fig. 8.4. A transition system \mathcal{A}

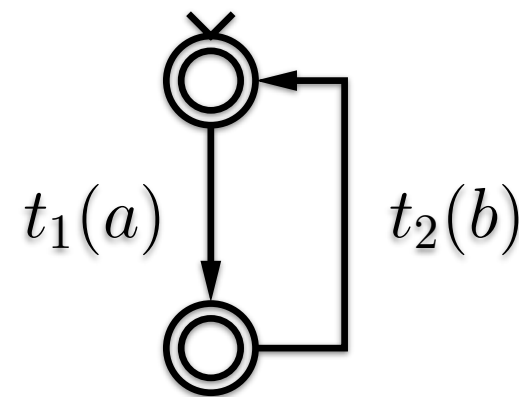


Fig. 8.5. A tester of the transition system of Fig. 8.4 for $\mathbf{G} (u_1 \Rightarrow \mathbf{X} \neg u_1)$

The full synchronization of the transition system of Fig. 8.4 and the tester of Fig. 8.5, namely $\mathcal{A} \parallel \mathcal{BT}_{\neg\psi_1}$, is in Fig. 8.6. The full synchronization $\mathcal{A} \parallel \mathcal{BT}_{\neg\psi_1}$ has an infinite history $(\langle a, t_1 \rangle \langle b, t_2 \rangle)^\omega$, and thus the infinite history $(ab)^\omega$ of \mathcal{A} violates ψ .

$$\mathbf{U} = \{ \mathbf{a} = \langle a, t_1 \rangle, \mathbf{b} = \langle b, t_2 \rangle \}$$

Proposition 8.17 allows us to reduce the model checking problem to the repeated executability problem. Recall that in the repeated executability problem we check if there exists an infinite history that executes global transitions from a set \mathbf{R} infinitely often. In contrast, in order to apply Prop. 8.17 we have to check if the history visits accepting states infinitely often. To reduce the latter to the former, it suffices to choose the set \mathbf{R} so that it contains the global transitions whose occurrence leads to an accepting state.

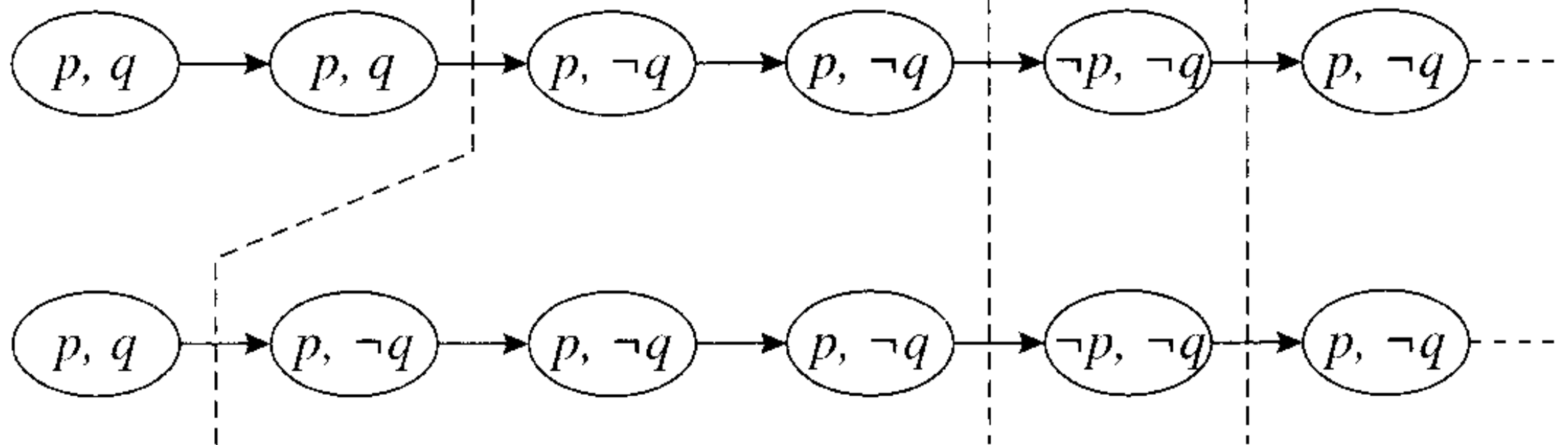
Theorem 8.19. *Let \mathbf{A} be a product and let ψ be a property of LTL. Let $\mathcal{BT}_{\neg\psi}$ be a tester for $\neg\psi$. Let \mathbf{R} be the set of global transitions $\langle \mathbf{t}, u \rangle$ of $\mathbf{A} \parallel \mathcal{BT}_{\neg\psi}$ such that the target state of u is an accepting state of $\mathcal{BT}_{\neg\psi}$. \mathbf{A} satisfies ψ if and only if the answer to the instance of the repeated executability problem given by $\mathbf{A} \parallel \mathcal{BT}_{\neg\psi}$ and \mathbf{R} is negative.*

While this approach is perfectly sensible when \mathcal{A} only has one component, it has a serious problem for general products when used with the unfolding method. To understand why, we take a closer look at the definition of full synchronization (Def. 8.16). Recall that $\langle \mathbf{t}, u \rangle$ if and only if $\mathbf{t} \in \mathbf{T}$, $u \neq \epsilon$, and $\mathbf{t} \in \lambda(u)$. In particular, it follows that $\mathcal{BT}_{\neg\psi}$ participates in *all* global transitions of $\mathbf{A} \parallel \mathcal{BT}_{\neg\psi}$. But, in this case, the final prefix of $\mathbf{A} \parallel \mathcal{BT}_{\neg\psi}$ does not contain any pair of concurrent events, and so the unfolding method does not present any advantage over exploring all the global states of the product.

The solution to the problem above is to replace the full synchronization by a “less intrusive” synchronization, in which the tester does not participate in all global transitions. This approach, which is the subject of this section, comes at a price (not very high, as we shall see): it only works for the stuttering-invariant fragment of LTL.

Definition 8.20. *Two infinite words $\pi, \pi' \in (2^{AP})^\omega$ are stuttering equivalent if there are two infinite sequences of positive integers $0 = i_0 < i_1 < i_2 < \dots$ and $0 = j_0 < j_1 < j_2 < \dots$ such that for every $k \geq 0$:*

$$\pi_{(i_k)} = \pi_{(i_k+1)} = \dots = \pi_{(i_{(k+1)}-1)} = \pi'_{(j_k)} = \pi'_{(j_k+1)} = \dots = \pi'_{(j_{(k+1)}-1)} .$$



A formula ψ of LTL is stuttering-invariant if $\pi \models \psi$ implies $\pi' \models \psi$ for every two stuttering equivalent words π, π' .

Example 8.21. Formulas like $\mathbf{F} p$ and $p \mathbf{U} q$ are stuttering-invariant. Loosely speaking, they promise that something will eventually happen, but do not specify when, which makes them insensitive to stuttering. On the contrary, the formula $\mathbf{X} p$ is not stuttering-invariant, since we have $\emptyset (\{p\})^\omega \models \mathbf{X} p$, but $\emptyset \emptyset (\{p\})^\omega \not\models \mathbf{X} p$.

Let us now consider our particular context, in which the set of atomic propositions AP is the set of local states of all components.

Definition 8.22. A global transition \mathbf{t} is stuttering w.r.t. ψ (or just stuttering, when ψ is clear from the context) if $AP_\psi \cap \bullet \mathbf{t} = AP_\psi \cap \mathbf{t}^\bullet$, where AP_ψ is the set of atomic propositions that occur in ψ .

Observe that if \mathbf{t} is stuttering then $\mathcal{S} = (\mathcal{S} \setminus \bullet\mathbf{t}) \cup \mathbf{t}\bullet$ holds for every $\mathcal{S} \subseteq AP_\psi$. Intuitively, the occurrence of a stuttering transition does not change the values of the atomic propositions that appear in ψ . The *non-stuttering projection* of a sequence σ of global transitions is the sequence obtained by removing all occurrences of stuttering transitions from σ .

The following proposition follows easily from the definitions:

Proposition 8.23. *Let ψ be a stuttering-invariant formula, and let σ, σ' be two infinite sequences of global transitions having the same non-stuttering projection. Then $\sigma \models \psi$ if and only if $\sigma' \models \psi$.*

We are now ready to define a new notion of synchronization between a tester for a stuttering-invariant property and a product.

Definition 8.24. Let $\mathcal{BT}_\psi = (\mathcal{B}_\psi, F_\psi, \lambda_\psi)$ be a tester for a stuttering-invariant formula ψ . The stuttering synchronization of \mathbf{A} and \mathcal{BT}_ψ is the product $\mathbf{A} \parallel^s \mathcal{BT}_\psi = \langle \mathcal{A}_1, \dots, \mathcal{A}_n, \mathcal{B}_\psi, \mathbf{U} \rangle$, where the synchronization constraint \mathbf{U} contains:

- a transition $\langle \mathbf{t}, u \rangle$ for every non-stuttering transition \mathbf{t} of \mathbf{A} and every transition u of \mathcal{BT}_ψ such that $\mathbf{t} = \lambda(u)$; and
- a transition $\langle \mathbf{t}, \epsilon \rangle$ for every stuttering transition \mathbf{t} of \mathbf{A} .

A transition $\langle \mathbf{t}, u \rangle$ of $\mathbf{A} \parallel^s \mathcal{BT}_\psi$ is stuttering if \mathbf{t} is stuttering or, equivalently, if $u = \epsilon$.

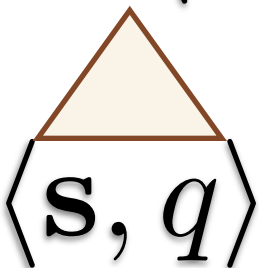
Intuitively, the tester in a stuttering synchronization only observes the non-stuttering transitions executed by the product. All others occur “silently”, without the knowledge of the tester. This motivates the following definition:

Definition 8.25. *An infinite history of \mathbf{A} is recurrent if it contains infinitely many occurrences of non-stuttering transitions. An infinite history of the stuttering synchronization $\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi}$ is recurrent if its projection on \mathbf{A} is recurrent or, equivalently, if the tester participates in infinitely many transitions.*

We will now show that Prop. 8.17 still holds for recurrent histories.

Proposition 8.26. *Let ψ be a stuttering-invariant formula of LTL and let $\mathcal{BT}_{\neg\psi}$ be a tester for $\neg\psi$. A recurrent infinite history of \mathbf{A} violates ψ if and only if it is the projection on \mathbf{A} of an infinite history of $\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi}$ that visits accepting states infinitely often.*

Consider now the case of non-recurrent histories. If an infinite history \mathbf{h}' of $\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi}$ is non-recurrent, then it can be split into a minimal length prefix \mathbf{h} and a suffix \mathbf{c} that starts at the smallest index i such that $\mathbf{c} = \langle \mathbf{t}_i, \epsilon \rangle \langle \mathbf{t}_{i+1}, \epsilon \rangle \langle \mathbf{t}_{i+2}, \epsilon \rangle \dots$ contains only stuttering transitions. Let $\langle \mathbf{s}, q \rangle$ be the global state reached by \mathbf{h}' after executing \mathbf{h} , i.e., right before the execution of $\langle \mathbf{t}_i, \epsilon \rangle$. The key intuition is that we can detect whether \mathbf{h}' violates ψ by looking at the tester's state q reached after executing \mathbf{h} . Prop. 8.28 proves that it suffices to check whether, loosely speaking, \mathcal{BT} with q as initial state recognizes some infinite sequence of \mathbf{A} containing only stuttering transitions. The intuition is that all such sequences are “indistinguishable”; if the tester recognizes any of them it will also recognize \mathbf{c} .

$$\mathbf{h}' = \mathbf{h} \langle \mathbf{t}_i, \epsilon \rangle \langle \mathbf{t}_{i+1}, \epsilon \rangle \langle \mathbf{t}_{i+2}, \epsilon \rangle \dots$$


Definition 8.27. A state q of \mathcal{BT} is **stutter-accepting** if some computation starting at q visits accepting states infinitely often and contains only transitions of \mathcal{BT} labeled by stuttering transitions of \mathbf{A} .

Proposition 8.28. Let ψ be a stuttering-invariant formula of LTL and let $\mathcal{BT}_{\neg\psi}$ be a tester for $\neg\psi$. A non-recurrent infinite history of \mathbf{A} violates ψ if and only if it is the projection on \mathbf{A} of an infinite history $\mathbf{h}' = \mathbf{h}\mathbf{c}$ (split into \mathbf{h} and \mathbf{c} as described above) of $\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi}$ satisfying the following properties:

- the tester's state reached by the occurrence of \mathbf{h} is stutter-accepting; and
- \mathbf{c} contains only stuttering transitions.

livelock is a history **h****t****c**

non-stuttering transitions

stuttering transitions

visible

invisible.

However, we still have to solve a small problem. If the initial state of the tester happens to be stutter-accepting, then in the history **h****c** the finite history **h** may be empty. In this case no **livelock monitor** leaves the tester in a stutter-accepting state, because the tester is in such a state from the very beginning! This technical problem can be solved by instrumenting $\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi}$:

Definition 8.29. *The instrumentation of $\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi}$, which we denote by $\mathcal{I}(\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi})$, is obtained by*

- *adding to each component \mathcal{A}_i of \mathbf{A} a new initial state is'_i and a new transition it_i leading from is'_i to the old initial state is_i ;*
- *adding to \mathcal{BT}_{ψ} a new initial state $is'_{\mathcal{BT}}$ and a new transition $it_{\mathcal{BT}}$ leading from $is'_{\mathcal{BT}}$ to the old initial state $is_{\mathcal{BT}}$; and*
- *adding to the synchronization constraint a new transition $\langle \mathbf{it}, it_{\mathcal{BT}} \rangle$, where $\mathbf{it} = \langle it_1, it_2, \dots, it_n \rangle$.*

Hauptergebnis:

Theorem 8.30. Let \mathbf{A} be a product and let ψ be a stuttering-invariant property of LTL. Let $\mathcal{BT}_{\neg\psi}$ be a tester for $\neg\psi$. Define:

- \mathbf{V} as the set containing the non-stuttering transitions of $\mathcal{I}(\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi})$ and the transition $\langle \mathbf{it}, it_{\mathcal{BT}} \rangle$; and
- \mathbf{R} and \mathbf{L} as the subsets of \mathbf{V} containing those transitions $\langle \mathbf{t}, u \rangle$ such that the **target state of u** is, respectively, an **accepting state** and a **stutter-accepting state** of $\mathcal{BT}_{\neg\psi}$.

\mathbf{A} satisfies ψ if and only if the answers to

- the instance of the **repeated executability problem** given by $\mathcal{I}(\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi})$ and \mathbf{R} ; and
- the instance of the **livelock problem** given by $\mathcal{I}(\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi})$, \mathbf{V} , and \mathbf{L} are both negative.

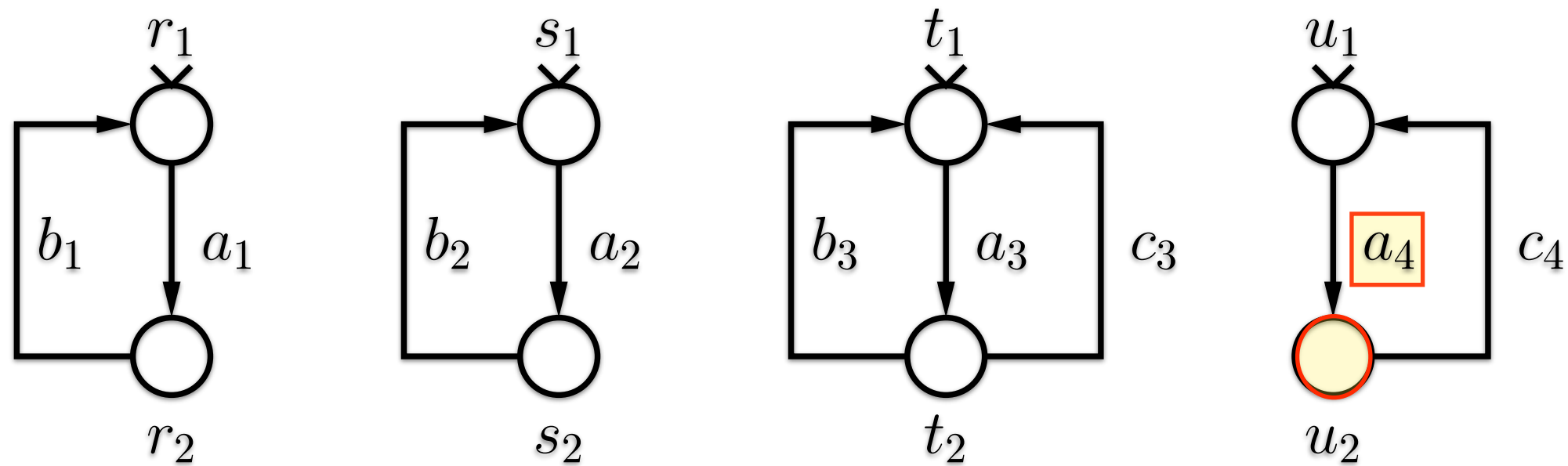
Proposition 8.31. *The set of stutter-accepting states of \mathcal{BT} can be computed in linear time in the size of product $\mathbf{A} \parallel^s \mathcal{BT}_{\neg\psi}$.*

Proof. Whether a given transition of \mathbf{A} is stuttering or not can be easily checked by inspecting its source and target states (Def. 8.22), and takes only linear time in the size of \mathbf{A} . Let \mathcal{BT}' be the result of removing from \mathcal{BT} all transitions u whose label $\lambda(u)$ is a non-stuttering transition of \mathbf{A} . By the definition of stutter-accepting, we have to compute the states q of \mathcal{BT}' such that some computation of \mathcal{BT}' starting at q visits accepting states infinitely often. A way to do this in linear time is to proceed in two steps:

- (1) compute the strongly connected components of \mathcal{BT}' that contain at least one accepting state and at least one edge whose both source and destination states belong to the component; and
- (2) return the states from which any of these components can be reached by a path of \mathcal{BT}' .

The algorithm is clearly correct. Step (1) can be performed in linear time using Tarjan's algorithm (see, e.g., [114]), while step (2) can be performed by means of a backward search starting at the states computed in (1). \square

Example 8.32. Consider the product of transition systems **A** in Fig. 8.7. We want to check whether the LTL formula $\psi = \mathbf{F G} (\neg u_2)$ holds in **A**. Since $AP_\psi = \{u_2\}$, a global transition **t** is stuttering if $\{u_2\} \cap \bullet \mathbf{t} = \{u_2\} \cap \mathbf{t}^\bullet$ (Def. 8.22). So the stuttering transitions are **a**₁, **a**₂, **a**₃, and **b**.



$$\mathbf{T} = \{ \mathbf{a}_1 = \langle a_1, \epsilon, \epsilon, \epsilon \rangle, \mathbf{a}_2 = \langle \epsilon, a_2, \epsilon, \epsilon \rangle, \mathbf{a}_3 = \langle \epsilon, \epsilon, a_3, \epsilon \rangle, \\ \mathbf{a}_4 = \langle \epsilon, \epsilon, \epsilon, a_4 \rangle, \mathbf{b} = \langle b_1, b_2, b_3, \epsilon, \epsilon \rangle, \mathbf{c} = \langle \epsilon, \epsilon, c_3, c_4 \rangle \}$$

Fig. 8.7. A product of transition systems **A** under LTL model checking

In order to check ψ we create a Büchi tester $\mathcal{BT}_{\neg\psi} = \mathcal{BT}_{\mathbf{GF}(u_2)}$, depicted in Fig. 8.8.³ In order to further simplify the figure we have used some conventions. By $d(\tau)$ we mean that the tester has four transitions $d(\mathbf{a}_1)$, $d(\mathbf{a}_2)$, $d(\mathbf{a}_3)$, and $d(\mathbf{b})$, labeled with \mathbf{a}_1 , \mathbf{a}_2 , \mathbf{a}_3 , and \mathbf{b} , respectively. Similarly for $f(\tau)$. The transitions $e(\mathbf{a}_4)$ and $g(\mathbf{a}_4)$ are labeled by \mathbf{a}_4 . Since the transitions $f(\tau)$ are labeled by stuttering transitions of the product, v_2 is also stutter-accepting. The tester accepts all infinite histories of \mathbf{A} which visit infinitely many global states at which u_2 holds.

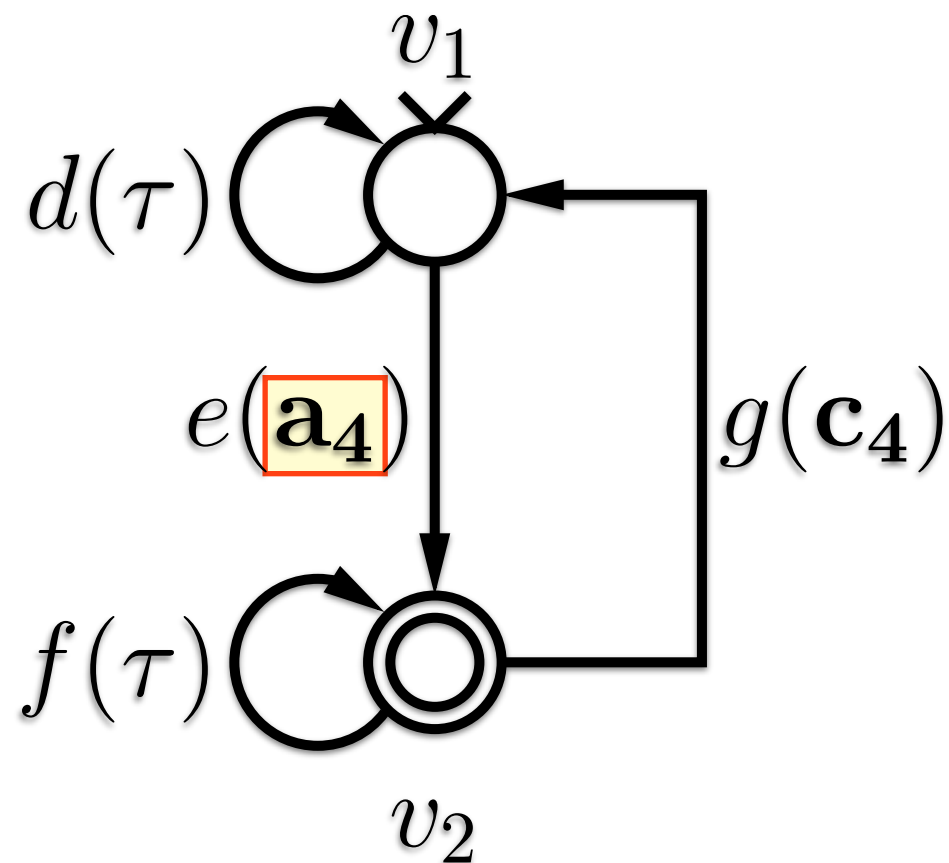
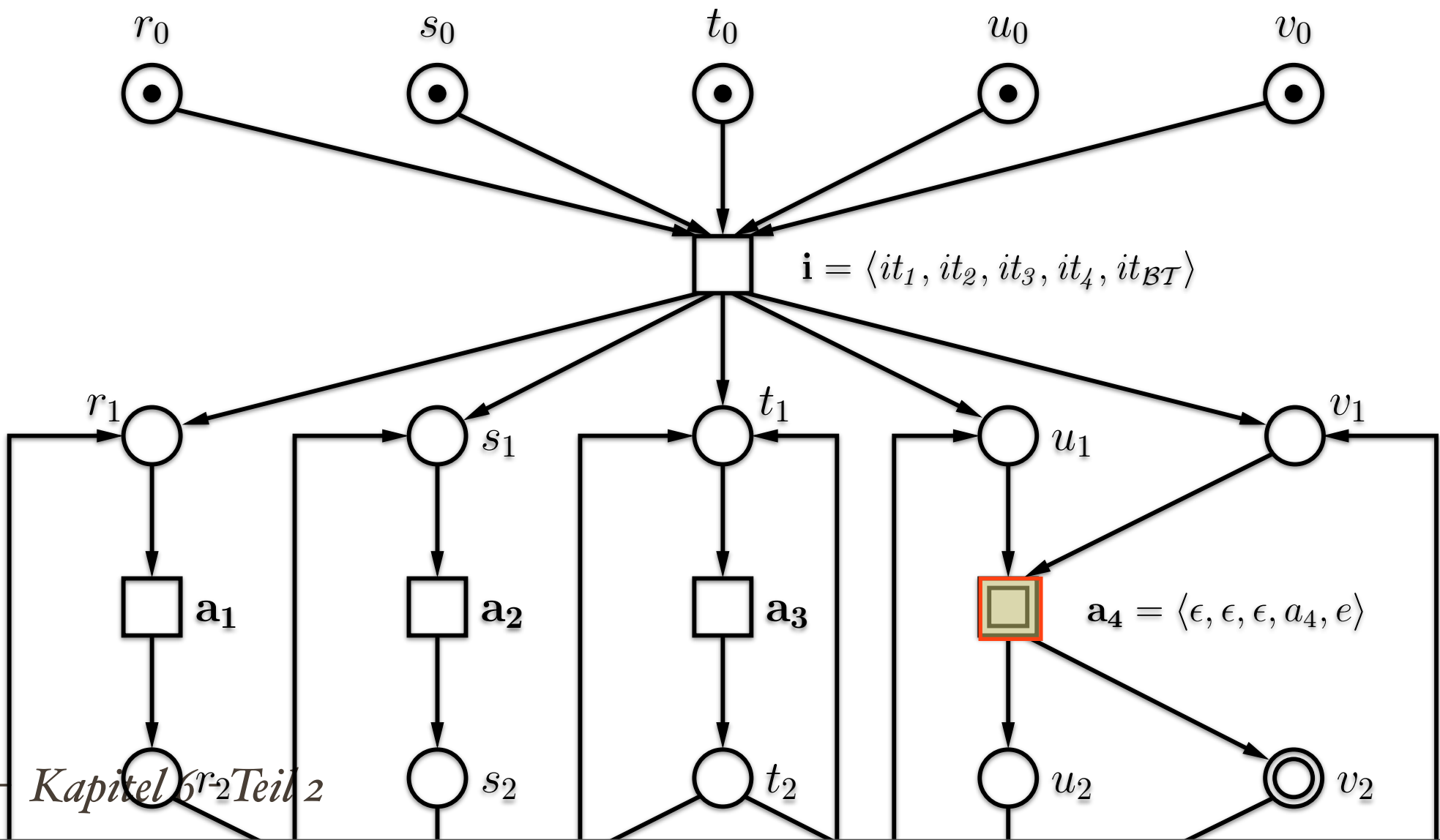


Fig. 8.8. Büchi tester of \mathbf{A} for $\mathbf{GF}(u_2)$

The stuttering synchronization of \mathbf{A} and $\mathcal{BT}_{\neg\psi}$ is a product \mathbf{P} , whose Petri net representation is shown in Fig. 8.9 (some global transitions which can never occur have been removed from \mathbf{P} to reduce clutter). The figure also shows the sets \mathbf{R} , \mathbf{V} , and \mathbf{L} defined in Thm. 8.30. The set \mathbf{V} contains the transition \mathbf{i} and the non-stuttering transitions of \mathbf{P} , i.e., \mathbf{a}_4 and \mathbf{c} . Since v_2 is the only accepting and stuttering-accepting state of the tester, the sets \mathbf{R} and \mathbf{L} coincide, and both contain the transition \mathbf{a}_4 . Graphically, we signal that \mathbf{a}_4 belongs to \mathbf{R} by drawing it with a double rectangle, and we signal that it belongs to \mathbf{L} by coloring it light grey.



Notice that the transitions d and f of the tester do not produce any transition in the stuttering synchronization. However, it is because of f that v_2 is stutter-accepting, and that is the reason why \mathbf{a}_4 is added to \mathbf{L} .

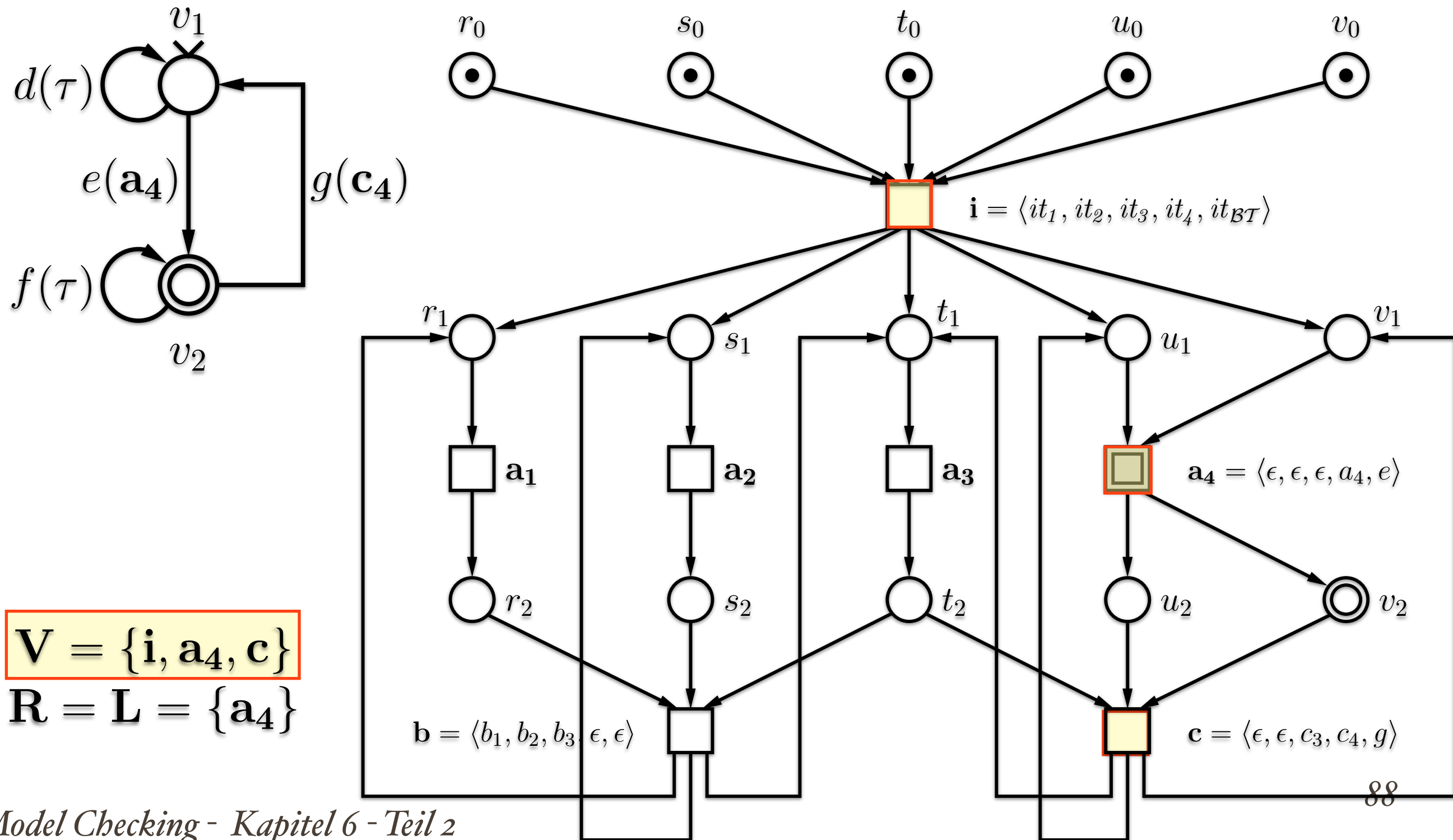
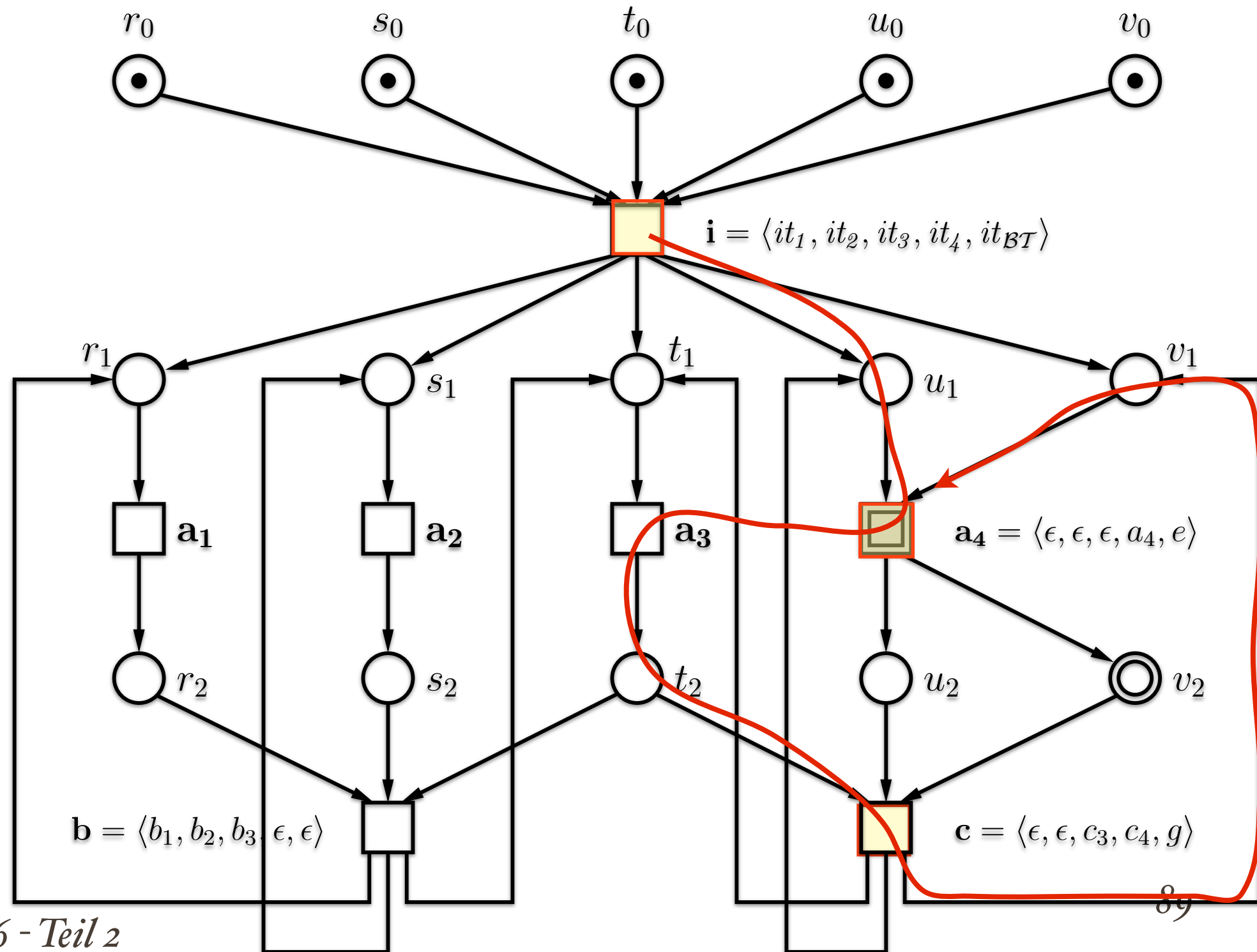
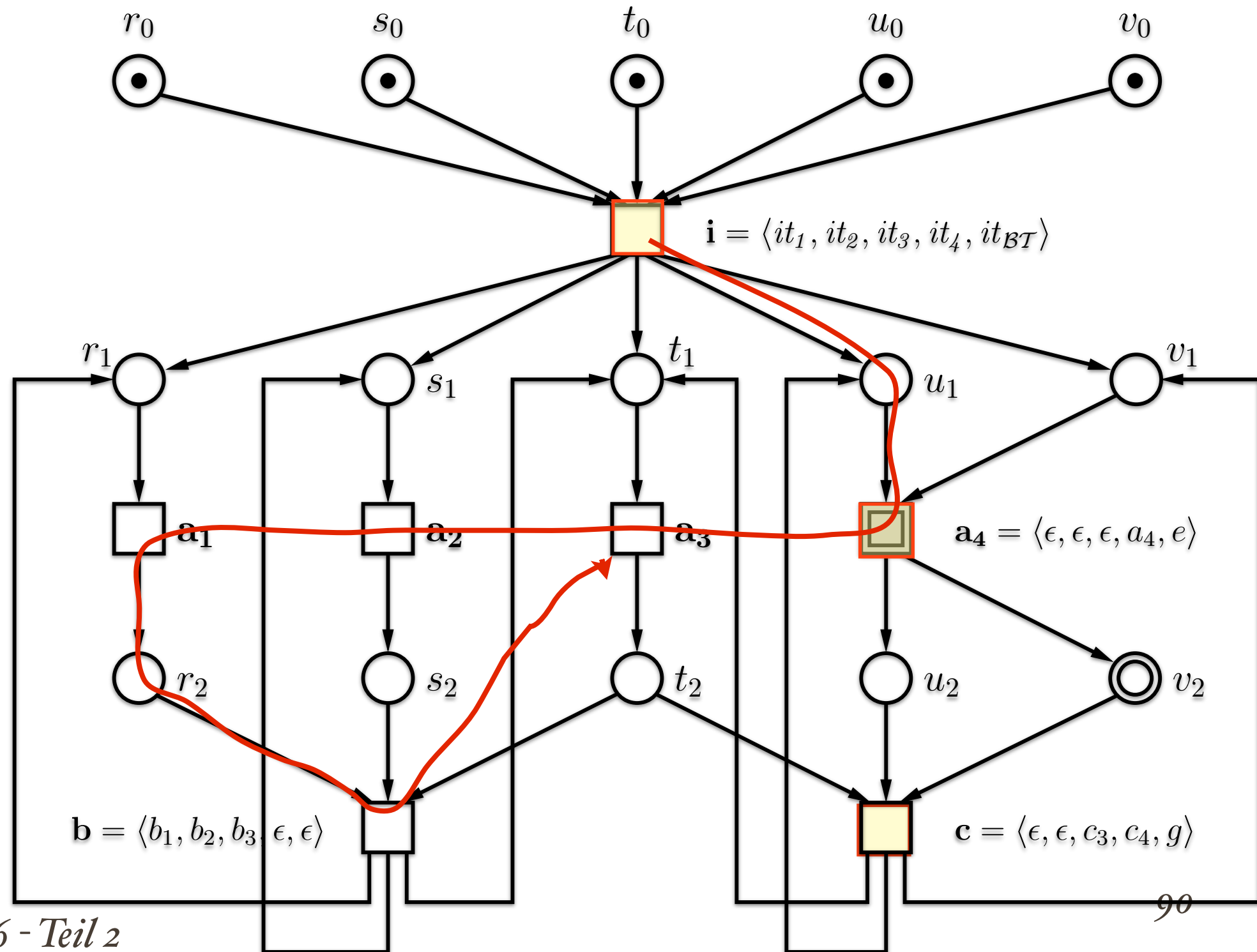


Figure 8.10 shows the final prefix for the repeated executability problem using the distributed size-lexicographic search strategy. Event 4 is a successful terminal having event 1 as companion. Terminal and companion correspond to the infinite global history $(\mathbf{a}_4 \mathbf{a}_3 \mathbf{c})^\omega$, which constitutes a counterexample to the property $\psi = \mathbf{F} \mathbf{G} (\neg u_2)$



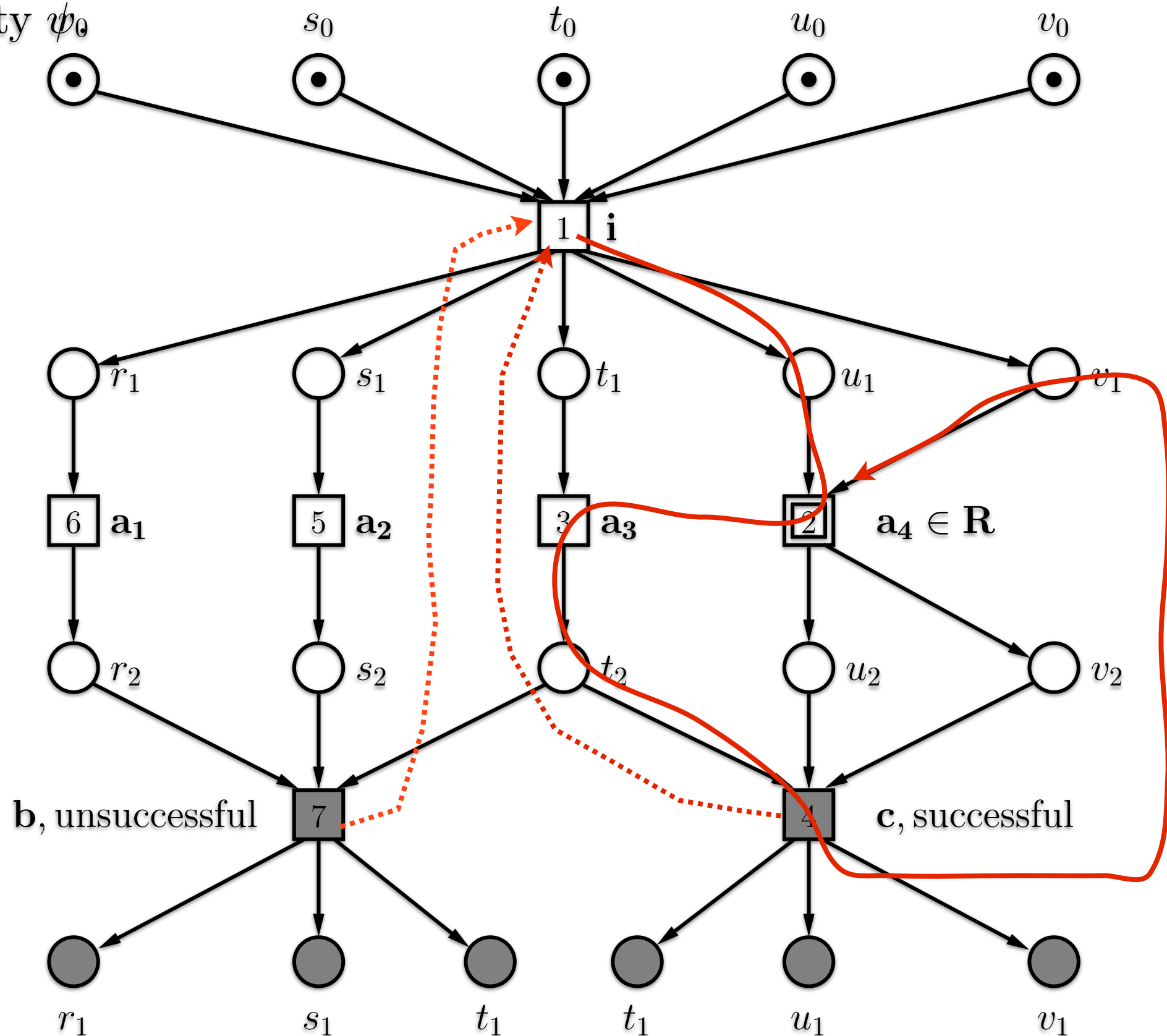
$\mathbf{V} = \{ \mathbf{i}, \mathbf{a}_4, \mathbf{c} \}$
 $\mathbf{R} = \mathbf{L} = \{ \mathbf{a}_4 \}$

Figure 8.10 shows the final prefix for the repeated executability problem using the distributed size-lexicographic search strategy. Event 4 is a successful terminal having event 1 as companion. Terminal and companion correspond to the infinite global history $\mathbf{a}_4(\mathbf{a}_3\mathbf{a}_2\mathbf{a}_1\mathbf{b})^\omega$, which constitutes a counterexample to the property $\psi = \mathbf{F}\mathbf{G}(\neg u_2)$

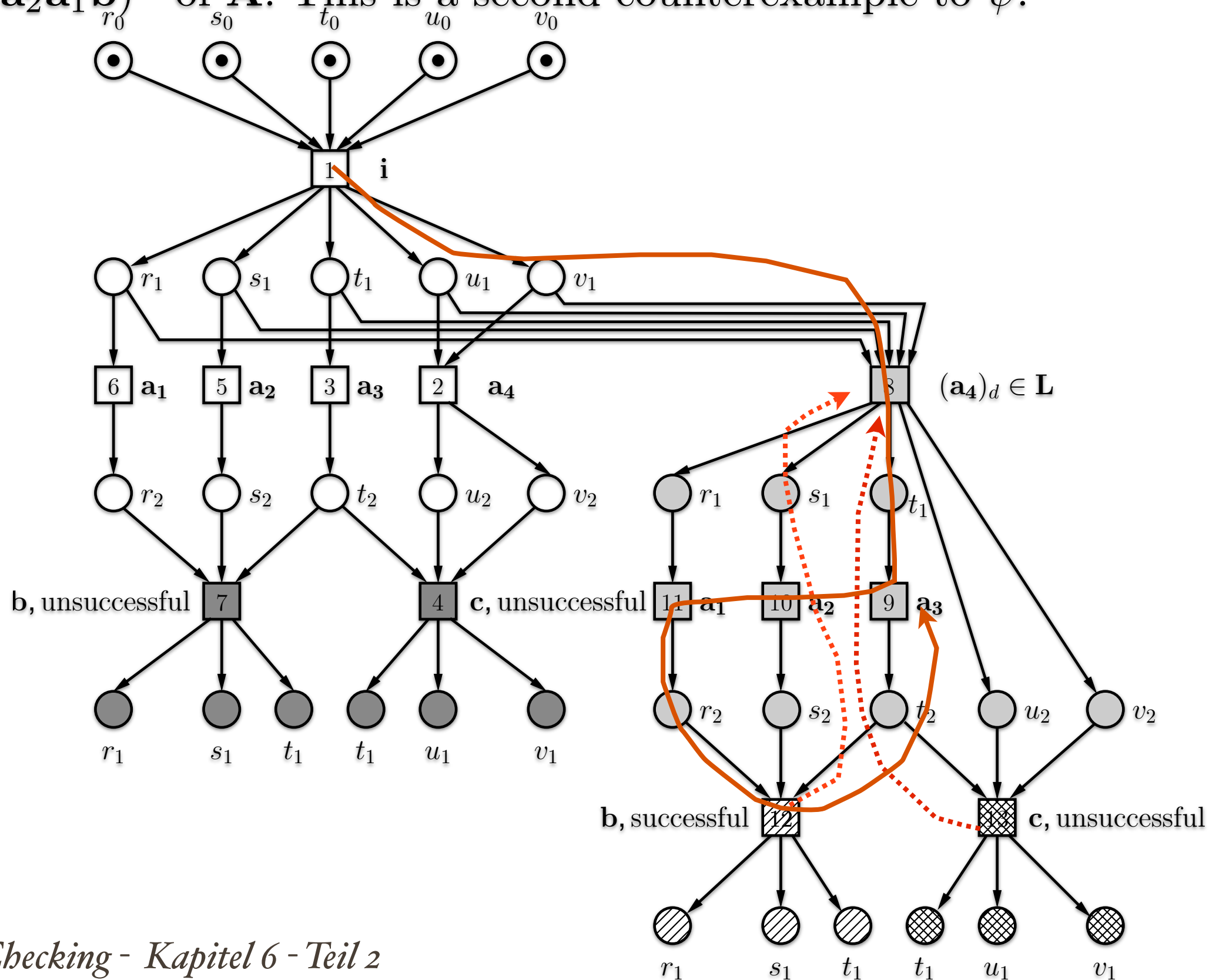


$\mathbf{V} = \{\mathbf{i}, \mathbf{a}_4, \mathbf{c}\}$
 $\mathbf{R} = \mathbf{L} = \{\mathbf{a}_4\}$

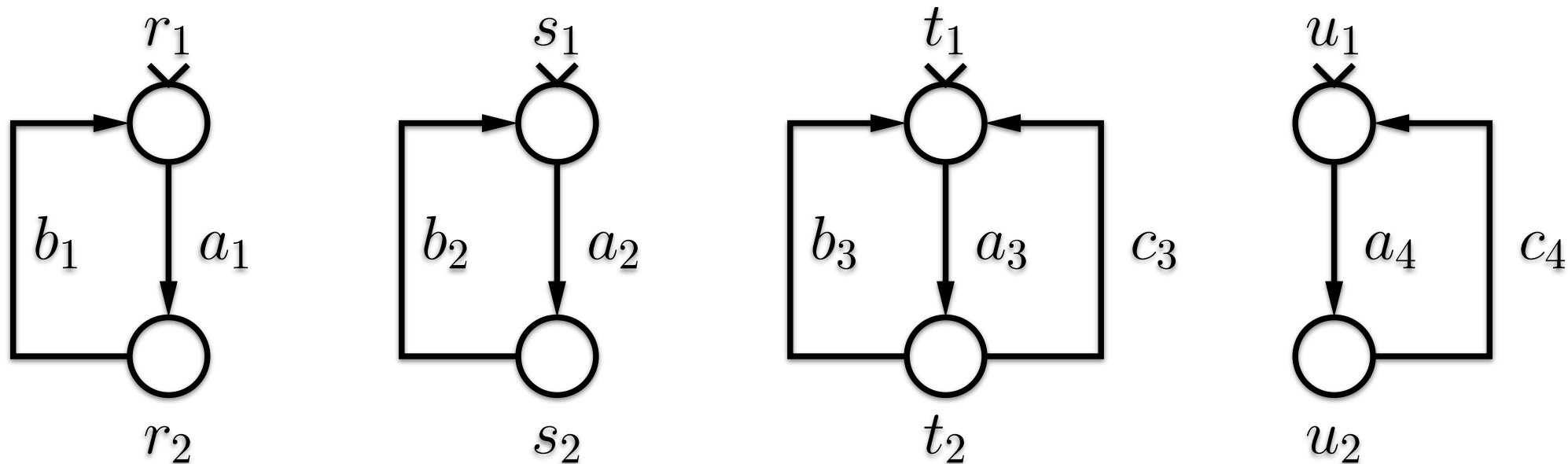
Figure 8.10 shows the final prefix for the repeated executability problem using the distributed size-lexicographic search strategy. Event 4 is a successful terminal having event 1 as companion. Terminal and companion correspond to the infinite global history $(\mathbf{a}_4\mathbf{a}_3\mathbf{c})^\omega$, which constitutes a counterexample to the property ψ_0



Similarly, Fig. 8.11 shows the final prefix for the livelock problem using a variant of the distributed size-lexicographic strategy. Also this prefix contains a successful terminal, corresponding to the infinite global history $\mathbf{a}_4(\mathbf{a}_3\mathbf{a}_2\mathbf{a}_1\mathbf{b})^\omega$ of \mathbf{A} . This is a second counterexample to ψ .



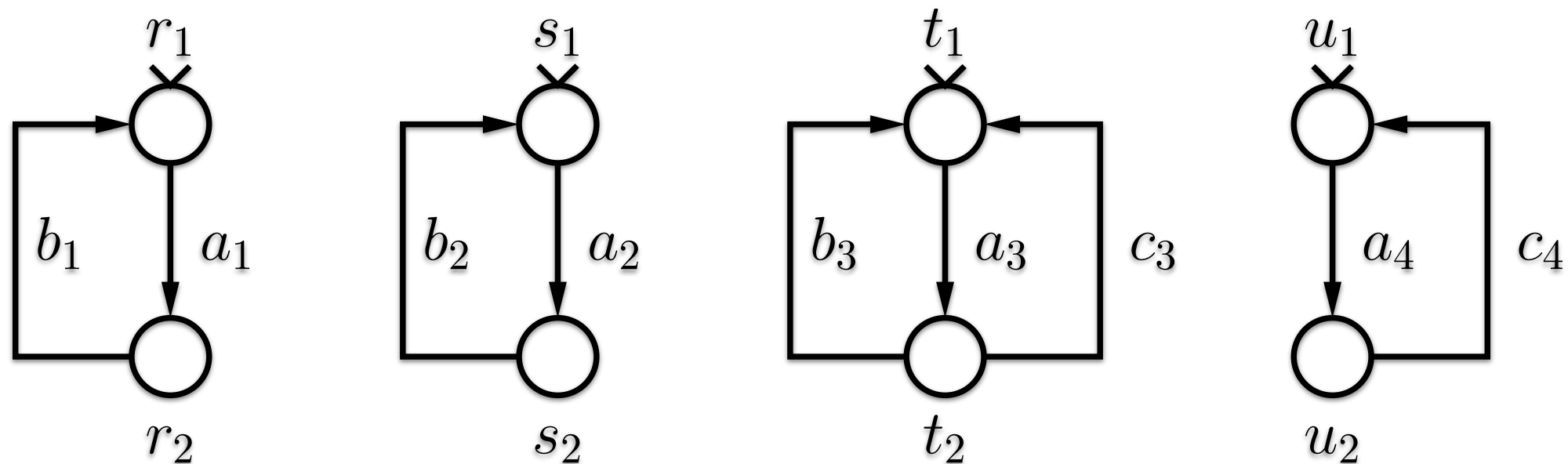
Consider now the family $\mathbf{A}_1, \mathbf{A}_2, \dots$ of products defined as follows. The product \mathbf{A}_n consists of $n - 2$ transition systems like the one on the left of Fig. 8.7 and the two transition systems on the right of the same figure. Its global transitions are: $\mathbf{a}_1, \dots, \mathbf{a}_n$, where the i th component of \mathbf{a}_i is a_i , and all other components are equal to ϵ ; $\mathbf{b} = \langle b_1, \dots, b_{n-2}, \epsilon, \epsilon \rangle$; and $\mathbf{c} = \{ \epsilon, \dots, \epsilon, c_{n-1}, c_n \}$.



$$\mathbf{T} = \{ \mathbf{a}_1 = \langle a_1, \epsilon, \epsilon, \epsilon \rangle, \mathbf{a}_2 = \langle \epsilon, a_2, \epsilon, \epsilon \rangle, \mathbf{a}_3 = \langle \epsilon, \epsilon, a_3, \epsilon \rangle, \\ \mathbf{a}_4 = \langle \epsilon, \epsilon, \epsilon, a_4 \rangle, \mathbf{b} = \langle b_1, b_2, b_3, \epsilon, \epsilon \rangle, \mathbf{c} = \langle \epsilon, \epsilon, c_3, c_4 \rangle \}$$

Fig. 8.7. A product of transition systems \mathbf{A} under LTL model checking

The product of Fig. 8.7 is the element of the family corresponding to $n = 2$. The transitions $\mathbf{a}_1, \dots, \mathbf{a}_n$ of \mathbf{A}_n are concurrent, and so \mathbf{A}_n has more than 2^n reachable global states. A model checking approach based on the (naive) exploration of the interleaving semantics can take **exponential time**. On the other hand it is easy to see that, whatever the strategy, the final prefixes for the repeated reachability and the livelock problems only grow **linearly in n** .



$$\mathbf{T} = \{ \mathbf{a}_1 = \langle a_1, \epsilon, \epsilon, \epsilon \rangle, \mathbf{a}_2 = \langle \epsilon, a_2, \epsilon, \epsilon \rangle, \mathbf{a}_3 = \langle \epsilon, \epsilon, a_3, \epsilon \rangle, \\ \mathbf{a}_4 = \langle \epsilon, \epsilon, \epsilon, a_4 \rangle, \mathbf{b} = \langle b_1, b_2, b_3, \epsilon, \epsilon \rangle, \mathbf{c} = \langle \epsilon, \epsilon, c_3, c_4 \rangle \}$$

Fig. 8.7. A product of transition systems \mathbf{A} under LTL model checking

9.2 Some Experiments

The material of this book attacks some questions of the theory of concurrency which we think have intrinsic interest. In particular, we have presented some fundamental results about which search strategies lead to correct search procedures. However, at least equally important is whether the theory leads to more efficient verification algorithms in terms of time, space, or both. This question must be answered experimentally, and in fact many of the papers mentioned in the last chapters contain experimental sections [39, 41, 42, 56, 57, 58, 60, 61, 70, 71, 74, 85, 84, 86, 87, 88, 109, 110, 121] in which the performance of the unfolding technique is measured and compared with the performance of other techniques. In particular, [110] compares the performance of PUNF, an implementation of a model checking algorithm similar to the one described in this book, and the Spin model checker.

Parallel LTL-X Model Checking of High-Level Petri Nets Based on Unfoldings

Claus Schröter¹ and Victor Khomenko²

¹ Institut für Formale Methoden der Informatik

Universität Stuttgart, Germany

`schroeter@fmi.uni-stuttgart.de`

² School of Computing Science

University of Newcastle upon Tyne, UK

`Victor.Khomenko@ncl.ac.uk`

Abstract. We present an unfolding-based approach to LTL-X model-checking of high-level Petri nets. It is based on the method proposed by Esparza and Heljanko for low-level nets [4, 5] and a state of the art parallel high-level net unfolders described in [15, 13]. We present experimental results comparing our approach to the one of [4, 5] and the model-checker SPIN [12].

R. Alur and D.A. Peled (Eds.): CAV 2004, LNCS 3114, pp. 109–121, 2004.
© Springer-Verlag Berlin Heidelberg 2004

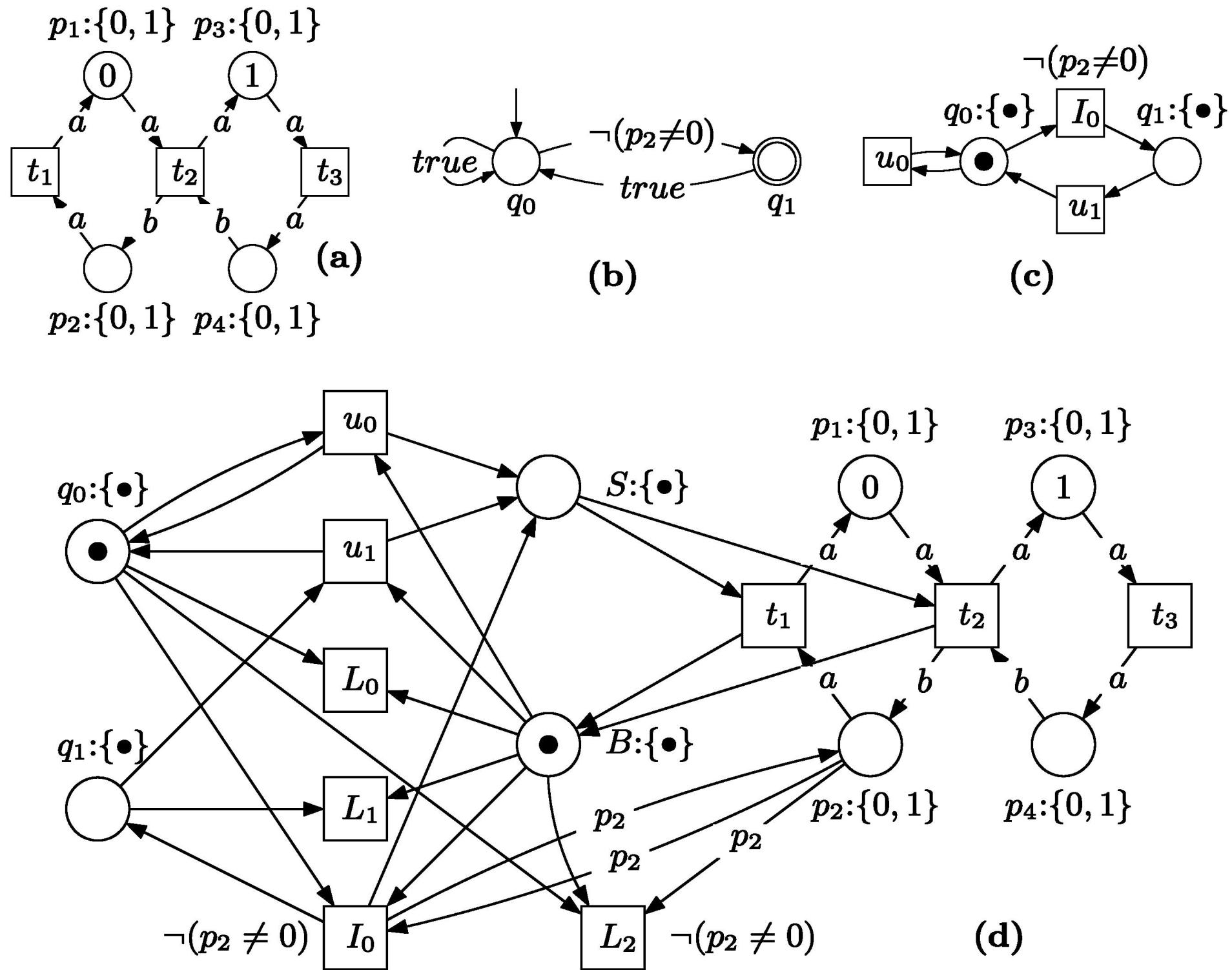


Fig. 2. M-net \mathcal{Y} : buffer of capacity 2 (a), Büchi automaton $A_{\neg\varphi}$ constructed for the property $\varphi \stackrel{\text{df}}{=} \diamond\Box(p_2 \neq 0)$ (b), the M-net corresponding to $A_{\neg\varphi}$ (c), and the product net $\mathcal{Y}_{\neg\varphi}$ (d).

Net	Formula	Result	UNFSMDLS	SPIN	PUNF
ABP	$\Box(p \rightarrow \Diamond q)$	True	0.19	0.01	0.08
BDS	$\Box(p \rightarrow \Diamond q)$	True	199	0.71	8.47
DPD(7)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	507	2.14	7.25
FURNACE(3)	$\Diamond \Box p$	True	1057	1.00	26.90
GASNQ(4)	$\Diamond \Box p$	True	240	0.14	8.46
RW(12)	$\Box(p \rightarrow \Diamond q)$	True	2770	0.44	47.67
FTP	$\Diamond \Box p$	True	>12000	3.99	836
OVER(5)	$\Diamond \Box p$	True	66.01	0.44	0.12
CYCLIC(12)	$\Box(p \rightarrow \Diamond q)$	True	0.38	11.25	0.08
RING(9)	$\Diamond \Box p$	True	2.13	1.64	0.13
DP(12)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	13.05	117	0.36
PH(12)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	0.04	0.61	0.02
COM(15,0)	$\Box(p \rightarrow \neg \Diamond q)$	True	—	3.11	0.02
PAR(5,10)	$\Box(p \rightarrow \neg \Diamond q)$	True	—	3.60	0.02

Net	SPIN	PUNF
CYCLIC(15)	168	0.08
CYCLIC(16)	478	0.07
CYCLIC(17)	1601	0.10
RING(12)	75.38	0.30
RING(13)	274	0.50
RING(14)	1267	0.85
DP(13)	559	0.53
DP(14)	2123	0.75
PH(15)	16.69	0.01
PH(18)	1570	0.01
COM(20,0)	232	0.02
COM(21,0)	686	0.03
COM(22,0)	2279	0.02
PAR(6,10)	161	0.02
PAR(7,10)	<i>mem</i>	0.04

Table 1(a) confirms that our approach outperforms UNFSMODELS on all examples, with an increase of speed up to 550 times (OVER(5) example). But we should mention that as noted in [5] UNFSMODELS is more or less an academic prototype, whereas PUNF is a high-performance unfolding engine. COM(15,0) and PAR(5,10) could not be verified with UNFSMODELS because either the Büchi automaton corresponding to the LTL-X formula for the low-level net (in the former case) or the low level net itself (in the latter case) could not be generated within reasonable time. Comparing PUNF and SPIN one can see that SPIN

Net	Formula	Result	UNFSMDLS	SPIN	PUNF
ABP	$\Box(p \rightarrow \Diamond q)$	True	0.19	0.01	0.08
BDS	$\Box(p \rightarrow \Diamond q)$	True	199	0.71	8.47
DPD(7)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	507	2.14	7.25
FURNACE(3)	$\Diamond \Box p$	True	1057	1.00	26.90
GASNQ(4)	$\Diamond \Box p$	True	240	0.14	8.46
RW(12)	$\Box(p \rightarrow \Diamond q)$	True	2770	0.44	47.67
FTP	$\Diamond \Box p$	True	>12000	3.99	836
OVER(5)	$\Diamond \Box p$	True	66.01	0.44	0.12
CYCLIC(12)	$\Box(p \rightarrow \Diamond q)$	True	0.38	11.25	0.08
RING(9)	$\Diamond \Box p$	True	2.13	1.64	0.13
DP(12)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	13.05	117	0.36
PH(12)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	0.04	0.61	0.02
COM(15,0)	$\Box(p \rightarrow \neg \Diamond q)$	True	—	3.11	0.02
PAR(5,10)	$\Box(p \rightarrow \neg \Diamond q)$	True	—	3.60	0.02

Net	SPIN	PUNF
CYCLIC(15)	168	0.08
CYCLIC(16)	478	0.07
CYCLIC(17)	1601	0.10
RING(12)	75.38	0.30
RING(13)	274	0.50
RING(14)	1267	0.85
DP(13)	559	0.53
DP(14)	2123	0.75
PH(15)	16.69	0.01
PH(18)	1570	0.01
COM(20,0)	232	0.02
COM(21,0)	686	0.03
COM(22,0)	2279	0.02
PAR(6,10)	161	0.02
PAR(7,10)	<i>mem</i>	0.04

performs better on the examples over the line. In contrast, PUNF outperforms SPIN on the examples under the line. We wanted to investigate this further and therefore we scaled up these systems and checked them again.

Net	Formula	Result	UNFSMDLS	SPIN	PUNF
ABP	$\Box(p \rightarrow \Diamond q)$	True	0.19	0.01	0.08
BDS	$\Box(p \rightarrow \Diamond q)$	True	199	0.71	8.47
DPD(7)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	507	2.14	7.25
FURNACE(3)	$\Diamond \Box p$	True	1057	1.00	26.90
GASNQ(4)	$\Diamond \Box p$	True	240	0.14	8.46
RW(12)	$\Box(p \rightarrow \Diamond q)$	True	2770	0.44	47.67
FTP	$\Diamond \Box p$	True	>12000	3.99	836
OVER(5)	$\Diamond \Box p$	True	66.01	0.44	0.12
CYCLIC(12)	$\Box(p \rightarrow \Diamond q)$	True	0.38	11.25	0.08
RING(9)	$\Diamond \Box p$	True	2.13	1.64	0.13
DP(12)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	13.05	117	0.36
PH(12)	$\Diamond \Box \neg(p \wedge q \wedge r)$	True	0.04	0.61	0.02
COM(15,0)	$\Box(p \rightarrow \neg \Diamond q)$	True	—	3.11	0.02
PAR(5,10)	$\Box(p \rightarrow \neg \Diamond q)$	True	—	3.60	0.02

Net	SPIN	PUNF
CYCLIC(15)	168	0.08
CYCLIC(16)	478	0.07
CYCLIC(17)	1601	0.10
RING(12)	75.38	0.30
RING(13)	274	0.50
RING(14)	1267	0.85
DP(13)	559	0.53
DP(14)	2123	0.75
PH(15)	16.69	0.01
PH(18)	1570	0.01
COM(20,0)	232	0.02
COM(21,0)	686	0.03
COM(22,0)	2279	0.02
PAR(6,10)	161	0.02
PAR(7,10)	<i>mem</i>	0.04

The results are shown in Table 1(b). They seem to confirm that SPIN's verification time grows exponentially with the size of the benchmark, whereas the verification time of PUNF remains less than a second in all cases. All these systems have a high degree of concurrency, and the results show that our partial order technique handles these systems quite well. In contrast, SPIN's partial order reductions are not very efficient on these examples.

Table 2. Experimental results for the parallel mode.

Net	SPIN	PUNF(1)	PUNF(2)
COM(20,3)	<i>mem</i>	8.58	6.01
COM(22,3)	<i>mem</i>	11.51	8.51
COM(25,3)	<i>mem</i>	17.29	12.84
PAR(20,100)	<i>mem</i>	8.60	4.84
PAR(20,150)	<i>mem</i>	31.98	18.28
BUF(20)	—	22.70	16.95
BUF(25)	—	142.72	89.40

As it was already mentioned, our unfolding routine supports multiple threads running in parallel. To demonstrate this we performed some experiments on a two processor machine.

The results are shown in Table 2. PUNF(n) means that PUNF makes use of n processors. The results confirm that the process of LTL-X model checking can be parallelised quite efficiently. This speeds up the verification, in the best case up to n times. Also the results show that these examples are not verifiable with SPIN because it runs out of memory. (We did not verify the BUF(n) system with SPIN because it was modelled directly as an M-net).

Inhalt

- *Kapitel 0:* Geschichte und Bedeutung des Model-Checking
- *Kapitel 1:* Die temporalen Logiken CTL und LTL
- *Kapitel 2:* Algorithmen für CTL und LTL
- *Kapitel 3:* Tools für CTL und LTL
- *Kapitel 4:* Binäre Entscheidungsbäume
- *Kapitel 5:* Symbolisches Model-Checking
- *Kapitel 6:* Model-Checking durch Auffalten