DOI 10.1007/s00165-007-0023-x
BCS © 2007
Formal Aspects of Computing (2007) 19: 475–485

Formal Aspects of Computing

Deciding orthogonal bisimulation

Thuy Duong Vu

Sectie Software Engineering, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands.
E-mail: tdvu@science.uva.nl

Abstract. Bergstra, Ponse and van der Zwaag introduced in 2003 the notion of *orthogonal bisimulation equivalence* on labeled transition systems. This equivalence is a refinement of branching bisimulation, in which consecutive tau's (silent steps) can be compressed into one (but not zero) tau's. The main advantage of orthogonal bisimulation is that it combines well with priorities. Here we solve the problem of deciding orthogonal bisimulation equivalence in finite (regular) labeled transition systems. Unlike as in branching bisimulation, in orthogonal bisimulation, cycles of silent steps cannot be eliminated. Hence, the algorithm of Groote and Vaandrager (1990) cannot be adapted easily. However, we show that it is still possible to decide orthogonal bisimulation with the same complexity as that of Groote and Vaandrager's algorithm. Thus if n is the number of states, and m the number of transitions then it takes $O(n(m+n))$ time to decide orthogonal bisimilarity on finite labeled transition systems, using $O(m+n)$ space.

Keywords: Concurrency theory; Orthogonal bisimulation equivalence; Branching bisimulation equivalence; Silent step; Labeled transition system

1. Introduction

Branching bisimulation equivalence proposed by van Glabbeek and Weijland [GW96] is a well-known and elegant equivalence in concurrency theory. This equivalence resembles, but is finer than the standard *observation equivalence* of Milner [Mil80].

In 2003, Bergstra et al. [BPZ03] introduced the notion of *orthogonal bisimulation equivalence* on labeled transition systems. Orthogonal bisimulation is a refinement of branching bisimulation in which consecutive τ-actions (silent steps) can be compressed into one (but not zero) τ-action. This is a major difference with branching bisimulation equivalence and other coarser semantics dealing with *abstraction* such as observation equivalence, delay bisimulation equivalence [Mil80, Mil81] and η-bisimulation equivalence [BG87].

The main advantage of orthogonal bisimulation, compared to branching bisimulation, is that it combines well with priorities [BPZ03]. Moreover, it has the following nice properties:

1. There is a modal logic based on Hennessy–Milner logic [HM85] which characterizes orthogonal bisimulation equivalence [BPZ03].
2. On closed terms in the setting of ACP (Algebra of Communicating Processes) with abstraction, orthogonal bisimulation congruence is completely axiomatized by three laws:

$$\begin{aligned} x\tau\tau &= x\tau \\ x\tau(y+z) &= x(y+z) \quad \text{if} \quad \tau y = \tau\tau y,\ \tau z = \tau\tau z \\ x(\tau(y+z)+z) &= x(y+z) \quad \text{if} \quad \tau y = \tau\tau y \end{aligned}$$

Correspondence and offprint requests to: T. D. Vu, E-mail: tdvu@science.uva.nl

We note that unlike in branching bisimulation equivalence, the axiom $x\tau = x$ is not sound in orthogonal bisimulation equivalence.

3. A trace characterization of orthogonal bisimulation equivalence, called the *compression structure of a process*, is provided in [Vu05]. The compression structure characterizes orthogonal bisimilarity in the same way as the *branching structure* characterizes branching bisimilarity in [Gla94].

A commonly used algorithm to analyze the complexity of branching bisimilarity on finite labeled transition systems is presented by Groote and Vaandrager [GV90]. This algorithm solves the *Relation Coarsest Partition with Stuttering problem* (RCPS) which is closely related to the *Relational Coarsest Partition problem* (RCP) [PT87]. It is shown in [GV90] that the algorithm for RCPS can be easily transformed to an $O(n(m+n))$ algorithm for deciding stuttering equivalence on finite Kripke structures [BCG88] and deciding branching bisimulation equivalence on finite labeled transition systems.

In this paper we take a step towards a theoretical foundation of orthogonal bisimulation by presenting an algorithm for deciding orthogonal bisimulation equivalence on finite labeled transition systems. This problem has been raised in [BPZ03]. Our approach is based on the work of Groote and Vaandrager. More precisely, the algorithm in this paper solves a generalization of the RCPS problem called *Relational Coarsest Partition with Stuttering problem characterizing Orthogonal bisimulation* (RCPSO), and therefore, can be used to decide orthogonal bisimulation. We note that in the Groote–Vaandrager algorithm, the authors perform a preprocessing step by eliminating the presence of cycles of silent steps. This is possible since if two states of a labeled transition system are strongly connected by silent steps, they are branching bisimilar. In the case of orthogonal bisimulation, we cannot eliminate the presence of cycles of silent steps. However, we show that the complexity of our algorithm remains the same as that of Groote and Vaandrager's algorithm. Thus, if n is the number of states and m the number of transitions, it takes $O(n(m+n))$ time and $O(n+m)$ space for deciding orthogonal bisimulation.

The structure of this paper is as follows. Section 2 recalls from [BPZ03] the definition of orthogonal bisimulation equivalence. Section 3 presents the RCPSO problem, and an algorithm to solve it. We show that this algorithm can be used to decide orthogonal bisimilarity. The paper is concluded with some remarks in Sect. 4.

2. Labeled transition systems and orthogonal bisimulation

In this section, we recall the definitions of labeled transition systems and orthogonal bisimulation from [BPZ03].

Definition 1 A *labeled transition system* (LTS) is a pair $(S, \rightarrow)$ with S a set of *processes* (or *states*), and $\rightarrow \subseteq S \times A \times S$ for a set A of *actions* (or *labels*) containing the *silent step* τ. A triple $(s, a, r) \in \rightarrow$ is called a *transition*.

An LTS is called *finite* if both S and A are finite.

We write $s \xrightarrow{a} r$ for $(s, a, r) \in \rightarrow$, $s \xrightarrow{a}$ for $\exists r \in S : s \xrightarrow{a} r$, and $s \xrightarrow{\tau} s'$ if there is a sequence $s_0 \dots s_n$ of states with $s_0 = s$, $s_n = s'$, $n \geqslant 0$, and $s_i \xrightarrow{\tau} s_{i+1}$ for all $i < n$. We note that $s \xrightarrow{\tau} s$ for all states $s \in S$. This is the case when $n = 0$.

Let τ-paths(s) be the set that consists of all sequences $s_0 \dots s_n$ of states with $s_0 = s$, $n \geqslant 0$, and $s_i \xrightarrow{\tau} s_{i+1}$ for all $i < n$.

Definition 2 (**Orthogonal bisimulation**). Let $(S, \rightarrow)$ be an LTS. An *orthogonal bisimulation* is a binary symmetric relation $\mathcal{B} \subseteq S \times S$ satisfying that for all states $s, r \in S$:

1. if $s\mathcal{B}r$ and $s \xrightarrow{a} s'$ for some s' and $a \neq \tau$, then $r \xrightarrow{a} r'$ for some r' with $s'\mathcal{B}r'$; and
2. if $s\mathcal{B}r$ and $s \xrightarrow{\tau} s'$ for some s', then $r \xrightarrow{\tau}$, and there is a path $r_0 \cdots r_n \in \tau$-paths(r) with $n \geqslant 0$ such that $s'\mathcal{B}r_n$ and $s\mathcal{B}r_i$ for all $i < n$.

Two states $s, r \in S$ are *orthogonally bisimilar*, denoted by $s \leftrightarrows_o r$, if there exists an orthogonal bisimulation $\mathcal{B}$ such that $s\mathcal{B}r$.

According to Definition 2, a state with a τ-outgoing transition will never be orthogonally bisimilar to a state without τ-outgoing transitions. Furthermore, the states of a cycle of silent steps are not orthogonally bisimilar in most cases. This is the reason why we cannot perform a preprocessing by eliminating cycles of silent steps as in [GV90]. Examples of orthogonal bisimulation are illustrated in Fig. 1.

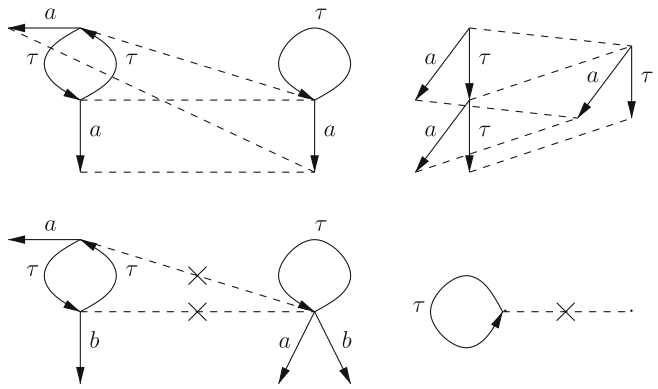

Fig. 1. Examples of orthogonal bisimulation. Here the dashed lines represent orthogonal bisimulation between processes

3. An efficient algorithm for deciding orthogonal bisimulation

In this section, we generalize the RCPS problem to the RCPSO problem that characterizes orthogonal bisimulation. Next, we will present an algorithm based on the algorithm in [GV90] to solve RCPSO. We also show that this algorithm can be used for deciding orthogonal bisimulation.

3.1. The RCPSO problem

We recall the definition of *partition* from [PT87, GV90] to describe RCPSO.

Definition 3 Let S be a set. A collection $\{B_i \mid i \in I\}$ of nonempty subsets of S is called a *partition* of S if $\cup_{i\in I} B_i = S$ and for $i \neq j : B_i \cap B_j = \emptyset$. The elements of a partition are called *blocks*. If P and P' are partitions of S then P' *refines* P (P is *coarser* than P') if any block of P' is included in a block of P. The *equivalence* $\sim_P$ on S induced by a partition P is defined by: $r \sim_P s$ if and only if $\exists\, B \in P$: $r \in B$ and $s \in B$.

The *Relational Coarsest Partition with Stuttering problem characterizing Orthogonal bisimulation* (RCPSO) can be specified as follows:
Given: a nonempty, finite set S of states, a relation $\rightarrow \subseteq S \times A \times S$ of *transitions* and an *initial partition* P_0 of S.
Find: the coarsest partition P_f satisfying:

1. P_f refines P_0;
2. if $s \sim_{P_f} r$ and $s \xrightarrow{a} s'$ with $a \neq \tau$, then there exists $r' \in S$ such that $r \xrightarrow{a} r'$ and $s' \sim_{P_f} r'$;
3. if $s \sim_{P_f} r$ and $s \xrightarrow{\tau} s'$, then there is an $n \geqslant 0$ and there are $r_0, \ldots, r_n \in S$ such that:
   (a) $r_0 = r$;
   (b) for all $0 \leqslant i < n$: $s \sim_{P_f} r_i$ and $r_i \xrightarrow{\tau} r_{i+1}$;
   (c) $s' \sim_{P_f} r_n$.

To decide orthogonal bisimulation, it is essential to start with a partition P_0 in which states with an outgoing τ-transition have been separated from states without an outgoing τ-transition. This agrees with orthogonal bisimulation equivalence.

3.2. The algorithm

This section describes an algorithm to solve the RCPSO problem. The algorithm is based on the algorithm for deciding branching bisimulation of Groote and Vaandrager [GV90], where transition systems might contain cycles of silent steps.

Let $\mid S \mid = n$ and $\mid \rightarrow \mid = m$. For blocks $B, B' \subseteq S$ we define $\mathrm{pos}_a(B, B')$ with $a \neq \tau$ as the set of states in B from which a state in B' can be reached by an *observable* action a. Furthermore, $\mathrm{pos}_\tau(B, B')$ is the set of states in B from which a state in B' can be reached by a sequence of silent steps τ.

$$\mathrm{pos}_a(B, B') = \{s \in B \mid \exists s' \in B' : s \xrightarrow{a} s'\} \quad \text{for } a \neq \tau,$$
$$\mathrm{pos}_\tau(B, B') = \{s \in B \mid \exists n \geqslant 0\, \exists s_0, \ldots, s_n : s_0 = s, \forall\ i < n : s_i \in B \wedge s_i \xrightarrow{\tau} s_{i+1} \text{ and } s_n \in B'\}.$$

Definition 4 We say that a block B' is a *splitter* of a block B *with respect to* a if and only if:

1. $B \neq B'$ or $a \neq \tau$, and
2. $\emptyset \neq \mathrm{pos}_a(B, B') \neq B$.

We note that Clause 1 in Definition 4 implies that in case $a = \tau$, a block B cannot be a splitter of itself.

If P is a partition of S and a block B' is a splitter of a block B with respect to a, then $\mathrm{Ref}_P^a(B, B')$ is the partition P where B is replaced by $\mathrm{pos}_a(B, B')$ and $B \setminus \mathrm{pos}_a(B, B')$.

Definition 5 A partition P is *stable with respect to a block* B' if for no block B of P and for no action a, B' is a splitter of B. P is *stable* if it is stable with respect to all its blocks.

The algorithm maintains a partition P that is initially P_0. It repeats the following steps until P is stable:

1. find blocks B, B' of P and a label $a \in A$ such that B' is a splitter of B with respect to a;
2. $P := \mathrm{Ref}_P^a(B, B')$.

Theorem 1 *The above algorithm for the RCPSO problem terminates after at most* $n - \mid P_0 \mid$ *refinement steps. The resulting partition* P_f *is the coarsest stable partition refining* P_0.

Proof. Sketch based on the proof of Theorem 3.1 in [GV90]. At each iteration of the refinement step, if we cannot find blocks B, B' of the current partition P and a label $a \in A$ such that B' is a splitter of B with respect to a then we know that the current partition is stable, and that the algorithm terminates. Otherwise, the number of blocks increases by one. Thus, termination will occur after at most $n - \mid P_0 \mid$ iterations. Next, we show that the resulting partition P_f is the coarsest stable partition refining P_0. We prove by induction on the number of refinement steps that any stable partition refining P_0 is also a refinement of the current partition P. Clearly the statement holds initially. Let R be a stable refinement of P_0. By the induction hypothesis, R is a refinement of P. Let Q be a refinement of P after a refinement step, using a splitting pair (B, B') with respect to a. We show that R is also a refinement of Q. Let C be a block in R. Then C is included in a block D of P. We prove that C is included in a block of Q. If $D \neq B$ then we are done. In the case $D = B$, we show that either $C \subseteq \mathrm{pos}_a(B, B')$ or $C \subseteq B \setminus \mathrm{pos}_a(B, B')$. Suppose that there are $r, s \in C$ with $s \in \mathrm{pos}_a(B, B')$ and $r \notin \mathrm{pos}_a(B, B')$. There are two cases:

1. $a \neq \tau$. There exists $s' \in B'$ such that $s \xrightarrow{a} s'$. Let C' be a block in R such that $s' \in C'$. Thus, $C' \subseteq B'$. Since R is a stable refinement of P_0 and $r, s \in C$, there exists $r' \in C' \subseteq B'$ such that $r \xrightarrow{a} r'$. This contradicts $r \notin \mathrm{pos}_a(B, B')$.
2. $a = \tau$. There are $s_0 = s, \ldots, s_n$ such that for all $i < n$: $s_i \in B$, $s_i \xrightarrow{\tau} s_{i+1}$ and $s_n \in B'$. Let $C_0 = C, \ldots, C_n$ be the blocks of R such that $s_i \in C_i$. Since R is a refinement of P and $s_n \in C_n \cap B'$ and for all $i < n$: $s_i \in C_i \cap B$, $C_i \subseteq B$ and $C_n \subseteq B'$. Since $s, r \in C$, there is a sequence $r_0, \ldots, r_m$ with $r_0 = r$, for all $i < m$, $r_i \in B$ and $r_i \xrightarrow{\tau} r_{i+1}$, and $r_m \in B'$. This contradicts $r \notin \mathrm{pos}_\tau(B, B')$.

Therefore, P_f is the coarsest stable partition refining P_0. □

We now describe how one can find in $O(m)$ time a splitter of the current partition, or find in $O(m)$ time that no such splitter exists. Furthermore, if a splitter has been found, it takes $O(m+n)$ time to refine the current partition. We will use the following definitions and lemmas.

Definition 6 Let P be a partition of S. A transition $s \xrightarrow{a} s'$ is called (P-)*inert* if $s \sim_P s'$ and $a = \tau$. A transition is *non-inert* if it is not an inert transition.

Definition 7 Let P be a partition of S. A (P-)*inert component* is a maximal subset $C \subseteq S$ such that for arbitrary states $s, s' \in C$ where $s \neq s'$ there is a path of inert transitions from s to s', and vice versa. Let B be a block of P such that $C \subseteq B \subseteq S$. We say that C is an inert component of B.

An inert component C of a block B is a *terminal component* of B if there is no inert component C' of B with $C' \neq C$ such that $s \xrightarrow{\tau} s'$ for some $s \in C$ and $s' \in C'$. An inert component of a block is called a *non-terminal component* if it is not a terminal component of that block.

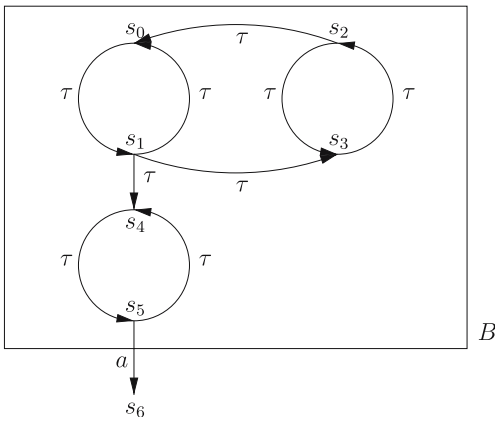

Fig. 2. An example of inert components

Note that an inert component can contain only one state (for example a state that is not connected by a τ-transition).

Example 1 Let B be a block consisting of states $s_0, s_1, s_2, s_3, s_4, s_5$, and a state $s_6 \notin B$ as illustrated in Fig. 2. Then the sets $C_1 = \{s_0, s_1, s_2, s_3\}$ and $C_2 = \{s_4, s_5\}$ are two inert components of B. More precisely, C_1 is a non-terminal component, while C_2 is a terminal component of B.

For a state s, an inert component C, a block B and an action a, we write $s \xrightarrow{a} B$ if there exists a state $s' \in B$ such that $s \xrightarrow{a} s'$, otherwise $s \xrightarrow{a} B$. Moreover, we write $C \xrightarrow{a} B$ if there exist states $s \in C$ and $s' \in B$ such that $s \xrightarrow{a} s'$, otherwise $C \xrightarrow{a} B$.

Lemma 1 *Let P be a refinement of P_0 and let $B, B' \in P$ and $a \in A$. Then B' is a splitter of B with respect to a if and only if:*

1. *$B \neq B'$ or $a \neq \tau$;*
2. *$r \xrightarrow{a} r'$ for some $r \in B$, $r' \in B'$;*
3. *if $a \neq \tau$ then there exists $s \in B$ such that $s \xrightarrow{a} B'$;*
4. *if $a = \tau$ then there exists a terminal component C of B such that $C \xrightarrow{\tau} B'$.*

Proof.

1. $\Rightarrow$: Suppose B' is a splitter of B. Clause 1 follows from Definition 4. Since $\mathrm{pos}_a(B, B') \neq \emptyset$ and $B \neq B'$ if $a = \tau$, there exists a transition $r \xrightarrow{a} r'$ for some $r \in B$, $r' \in B'$. Clause 3 follows from the fact that $\mathrm{pos}_a(B, B') \neq B$. To prove Clause 4, we assume that for every terminal component C of B there is a state $s' \in B'$ such that $s \xrightarrow{\tau} s'$ for some $s \in C$. It follows from Definition 7 that every state in B can lead to a state in B' by a sequence of τ transitions. Thus, $\mathrm{pos}_a(B, B') = B$. This is a contradiction to the fact that B' is a splitter of B.
2. $\Leftarrow$: Suppose that B and B' satisfy Clause 1, 2, 3 and 4. Then B' is a splitter of B because:
   (a) $\emptyset \neq \mathrm{pos}_a(B, B')$ because of Clause 2.
   (b) $\mathrm{pos}_a(B, B') \neq B$ because of Clause 3 and Clause 4. □

Example 2 Let B_0, B_1, B_2 and B_3 be the blocks given in Fig. 3. Then B_1 is a splitter of B_0 with respect to a, and B_3 is a splitter of B_2 with respect to τ. More precisely, $\mathrm{pos}_a(B_0, B_1) = \{s_0, s_2\}$ and $\mathrm{pos}_\tau(B_2, B_3) = \{s_6\}$.

We note that in the case of branching bisimulation, each state is an inert component since the initial P_0 does not have cycles of τ-transitions. Therefore, instead of dealing with terminal-components, one has to deal with *bottom-states* only. This is the main difference between the Groote–Vaandrager algorithm and our algorithm. Moreover, while the initial partition of Groote–Vaandrager consists of a single block containing all states, our initial partition will consist of two blocks: one block of states that can perform a τ-transition, and one block of states that cannot perform a τ-transition.

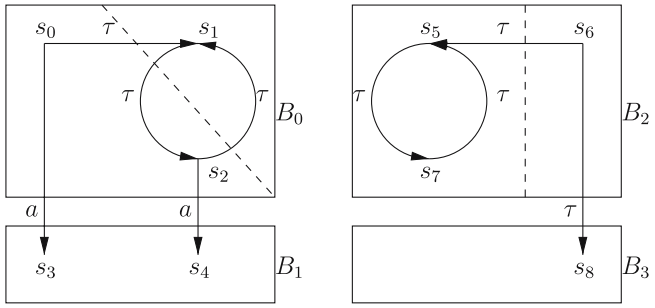

Fig. 3. An example of splitting

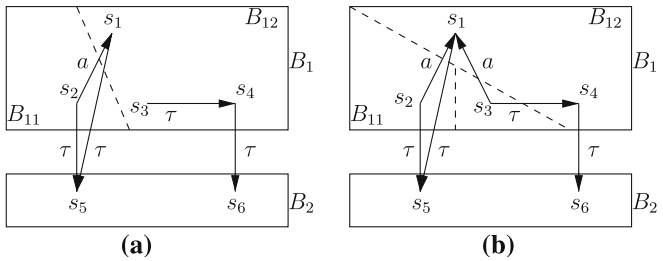

Fig. 4. An example of stability **(a)** and unstability **(b)**

Lemma 2 *Let P and R be partitions such that R refines P, and P and R have the same inert transitions. Let B' be a block of both P and R such that P is stable with respect to B'. Then R is stable with respect to B'.*

Proof. We prove this lemma by contradiction. Suppose that there exists a block B of R and an action a such that B' is a splitter of B with respect to a. There are two cases:

1. $a \neq \tau$. By Lemma 1, there exists a transition $r \xrightarrow{a} r'$ with $r \in B$, $r' \in B'$, and a state $s \in B$ such that for no $s' \in B'$: $s \xrightarrow{a} s'$. Since R refines P, B is included in a block B'' of P. Thus, $r, s \in B''$. By Lemma 1, B' is a splitter of B''. This contradicts the fact that P is stable with respect to B'.
2. $a = \tau$. Then $B \neq B'$. By Lemma 1, there is a terminal component C of B such that for no $s' \in B'$: $s \xrightarrow{\tau} s'$ for some $s \in C$. Since R refines P, B is included in a block B'' of P. Thus C is included in an inert component C'' of B''. We prove that C'' is also a terminal component of B''. Suppose that C'' is not a terminal component of B''. Then there exists an inert component K with $K \neq C''$, and an inert transition $r \xrightarrow{\tau} r'$ of B'' with $r \in C''$ and $r' \in K$. Let $p \in C$. Then $p \in C''$. Thus there is a path $p_0 \ldots p_n$ of states such that $p_0 = p$ and $p_n = r$ with $p_i \sim_P p_{i+1}$ for all $i < n$. Since P and R have the same inert transitions, $p_i \sim_R p_{i+1}$ for all $i < n$, and $r \sim_R r'$. It follows from Definition 3 and $p \in B$ that $r' \in B$. Since C is a terminal component of B and Definition 7, $r' \in C$. Thus, $r' \in C''$. This contradicts the fact that $r' \in K$. □

Example 3 Let $(S, \rightarrow)$ be the LTS illustrated in Fig. 4a.

$$\begin{aligned} S &= \{s_1, s_2, s_3, s_4, s_5, s_6\} \text{ and} \\ \rightarrow &= \{s_1 \xrightarrow{\tau} s_5, s_2 \xrightarrow{a} s_1, s_2 \xrightarrow{\tau} s_5, s_3 \xrightarrow{\tau} s_4, s_4 \xrightarrow{\tau} s_6\}. \end{aligned}$$

Let $B_1 = \{s_1, s_2, s_3, s_4\}$, $B_2 = \{s_5, s_6\}$ and $P = \{B_1, B_2\}$. It is obvious that P is stable with respect to B_2. Moreover, B_1 is a splitter of itself with respect to a. We split B_1 into $B_{11} = \{s_2\}$ and $B_{12} = \{s_1, s_3, s_4\}$. Let R be the refinement, $R = \{B_{11}, B_{12}, B_2\}$. Then P and R have the same inert transition $s_3 \xrightarrow{\tau} s_4$. Therefore, R is also stable with respect to B_2.

We now extend $(S, \rightarrow)$ with a transition $s_3 \xrightarrow{a} s_1$ (see Fig. 4b). The partition $P = \{B_1, B_2\}$ is still stable with respect to B_2. Furthermore, B_1 is also a splitter of itself with respect to a. However, after the splitting of B_1 into

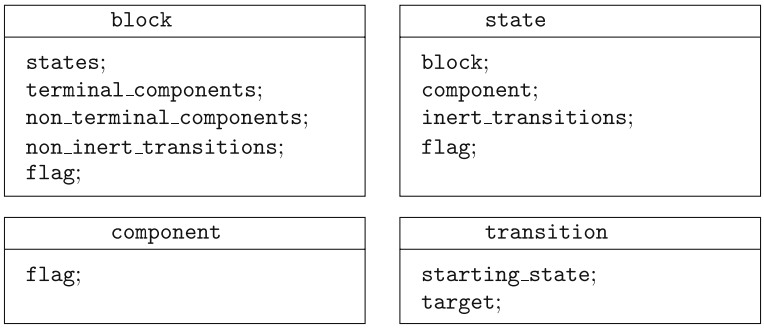

Fig. 5. Data structures for solving RCPSO

$B_{11} = \{s_2, s_3\}$ and $B_{12} = \{s_1, s_4\}$, the inert transition $s_3 \xrightarrow{\tau} s_4$ becomes non-inert. The refinement R is no longer stable with respect to B_2 as B_2 can be used as a splitter of B_{11} with respect to τ ($\text{pos}_\tau(B_{11}, B_2) = \{s_2\}$).

Given an LTS, the data structure for an implementation for solving the RCPSO problem is initialized as follows, where we identify a block, a component and a state with a record representing it (transitions are represented indirectly):

- There are two lists of blocks tobeprocessed and stable. A block B' is in stable if the current partition is stable with respect to B', otherwise B' is in tobeprocessed. Initially, all blocks in P_0 are in the list tobeprocessed.
- Each state contains two pointers block and component to the block and the inert component of which it is an element, and a list inert_transitions of inert transitions ending in this state (see Fig. 5).
- Each block B contains a list states of states in B.
  Furthermore, it has a list `terminal_components` of terminal components in B and a list `non_terminal_components` of non-terminal components in B. Finally, it points to a list non_inert_transitions of groups of non-inert transitions that end in B. More precisely, all transitions with the same label are in subsequent records in the list. To compute the lists of terminal and non-terminal components of all blocks in the initial partition, one can apply a variant of the standard depth first search algorithm [AHU74] using $O(m + n)$ time and space. In addition, grouping of the non-inert transitions has a complexity $O(|A| + m)$ (bucket sort).
- Each transition contains two pointers starting_state and target: one to its starting state, and one to its target.
- Each state, each component and each block has an auxiliary field flag of type boolean, which is 0 (standing for false) initially.
- Moreover, there are two auxiliary booleans found_a_splitter and inert_becomes_non_inert, and an auxiliary list BL. Initially, found_a_splitter = false, inert_becomes_non_inert = false and BL = $\emptyset$. We note that given a block B', the block list BL contains all blocks B having a non-inert transition from B to B'.

Note that the transitions of the LTS are either represented in the blocks (the non-inert ones), or in the states (the inert ones).

The implementation of the algorithm for deciding the RCPSO problem is presented in Tables 1 and 2.

With reference to Table 1, we first explain how to compute in $O(m)$ time whether we can find a splitter of the current partition or decide that no such splitter exists, meaning that the current partition is stable. Let $m^a_{\mathtt{BB}'}$ denote the number of transitions from a block B to a block B' with label a. Let B' be a block in tobeprocessed. Scan the list L of groups of non-inert transitions that end in B' (initially, $L = B'$.non_inert_transitions). Consider subsequently all groups T_a of non-inert transitions with a label a in L. We set the flag field of the blocks of the starting states of all transitions in T_a, and add these blocks to the list BL. Furthermore, if a is an observable action then we raise the flag of the starting states of all transitions in T_a. In this case, to find out whether B' is a splitter of a block B in BL, we only have to check whether the flag of some state in B is not raised. In case $a = \tau$, we raise the flag of the components of the starting states of all transitions in T_τ. To find out whether B' is a splitter of a block B in BL with respect to τ, we only have to check whether the flag of some terminal-component in B is not raised. The complexity to find out that B' is a splitter of B with respect to an action a or not is $O(m^a_{\mathtt{BB}'})$. Therefore, the complexity to find a splitter of the current partition (if it exists) is $\Sigma O(m^a_{\mathtt{BB}'})$ or $O(m)$.

Table 1. The algorithm for solving the RCPSO problem

```
(1) tobeprocessed = P_0; stable = ∅;
(2) repeat
(3)         Let B' = head(tobeprocessed);
(4)         // Scan the list L of all non-inert transitions that end in B'
(5)          L = B'.non_inert_transitions;
(6)         if  L ≠ ∅ then repeat
(7)                     Let T_a = head(L); L = tail(L);
(8)                     BL = ∅;
(9)                     for all transitions s -a-> s' ∈ T_a do
(10)                            if s.block.flag = 0 then s.block.flag = 1; insert(s.block, BL); end if;
(11)                            case a ≠ τ: s.flag = 1;
(12)                            case a = τ: s.component.flag = 1;
(13)                            end case;
(14)                    end for;
(15)                    repeat
(16)                            Let B = head(BL); BL = tail(BL);
(17)                            case a ≠ τ:
(18)                            if there is a state s ∈ B such that s.flag = 0 then
(19)                                    found_a_splitter = true;
(20)                            case a = τ:
(21)                            if there is a terminal-component C ∈ B such that C.flag = 0 then
(22)                                    found_a_splitter = true;
(23)                            end case;
(24)                    until found_a_splitter or BL = ∅;
(25)                    if not found_a_splitter then Reset all flags;
(26)            until found_a_splitter or L = ∅;
(27)            end if ;
(28)            if found_a_splitter then
(29)                    B_1, B_2 = split(B, a);
(30)                    tobeprocessed = remove(B, tobeprocessed);
(31)                    tobeprocessed = insert(B_1, insert(B_2, tobeprocessed));
(32)                    if  inert_becomes_non_inert then
(33)                            tobeprocessed = tobeprocessed ∪ stable; stable = ∅;
(34)                    end if;
(35)                    Reset all flags;
(36)                    found_a_splitter = false; inert_becomes_non_inert = false;
(37)            else
(38)                    tobeprocessed = remove(B', tobeprocessed); stable = insert(B', stable);
(39)            end if;
(40) until tobeprocessed = ∅
```

The functions insert, head and tail are standard functions on lists. The function remove denotes removal of an element from a list, and the function ∪ denotes concatenation of two lists

In the case we have found that B' is a splitter of a block B in the current partition, we split B into B_1 and B_2 and insert these blocks to the list tobeprocessed. By Lemma 2, if some inert transition of the current partition becomes a non-inert transition in the new partition then we append the list stable to the list tobeprocessed and make stable empty. If B' is not a splitter in the current partition, then we move B' from the list tobeprocessed to the list stable, and repeat the same procedure for the next block in tobeprocessed. If tobeprocessed is empty then we know that the current partition is stable.

With reference to Table 2, we now explain how to split B by B' into B_1 and B_2 in $O(m_B + n_B)$ time, where m_B is the number of transitions and n_B the number of states in B. In case a is a τ-action, we raise the flag of all states in B that can lead to a state in a terminal-component with a raised flag by a path of inert transitions. To do this, one can apply a standard depth first search algorithm using $O(m_B + n_B)$ time and space. (Here we use the list of inert transitions ending in each state of B). We now refine B into B_1 and B_2. All states with a raised flag are inserted into B_1, the others are placed in B_2. It is easy to compute the list of non-inert transitions ending in B_1 and B_2, and the lists of inert transitions ending in each state of B_1 and B_2 (line 11–35 of Table 2). Since the set of actions is finite, one can apply the bucket sort algorithm [AHU74] to group the non-inert transitions of B_1 and B_2 in $O(m_B)$ time. Finally, one can apply the well-known algorithm for finding strongly connected components in a directed graph [AHU74] using $O(m_B + n_B)$ time and space to compute the lists of terminal and non-terminal components for B_1 and B_2.

Table 2. How to construct B_1 and B_2

```
split(B, a)
(1)     if a = τ then
(2)         Raise the flag of all states in B that can lead to a state in a
(3)         terminal-component with a raised flag by a path of inert transitions;
(4)     end if;
(5)     B1 = new; B2 = new;
(6)     // Assign the states to B1 and B2
(7)     for all states s ∈ B.states do
(8)         if s.flag = 1 then insert(s, B1.states)
(9)             else insert(s, B2.states); end if;
(10)    end for;
(11)    //Compute the list of non-inert transitions of B1 and B2
(12)    for all transitions t ∈ B.non_inert_transitions do
(13)        if t.target ∈ B1 then insert(t, B1.non_inert_transitions);
(14)        else insert(t, B2.non_inert_transitions);end if;
(15)    end for;
(16)    //Compute the list of inert transitions ending in each state of B1
(17)    for all states s ∈ B1.states do
(18)        for all transitions t ∈ s.inert_transitions do
(19)            if t.starting_state ∉ B1.states then
(20)                inert_becomes_non_inert = true;
(21)                remove(t, s.inert_transitions);
(22)                insert(t, B1.non_inert_transitions);
(23)            end if;
(24)        end for;
(25)    end for;
(26)    //Compute the list of inert transitions ending in each state of B2
(27)    for all states s ∈ B2.states do
(28)        for all transitions t ∈ s.inert_transitions do
(29)            if t.starting_state ∉ B2.states then
(30)                inert_becomes_non_inert = true;
(31)                remove(t, s.inert_transitions);
(32)                insert(t, B2.non_inert_transitions);
(33)            end if;
(34)        end for;
(35)    end for;
(36)    Group non-inert transitions of B1 and B2;
(37)    Compute the lists of terminal and non-terminal components for B1 and B2;
```

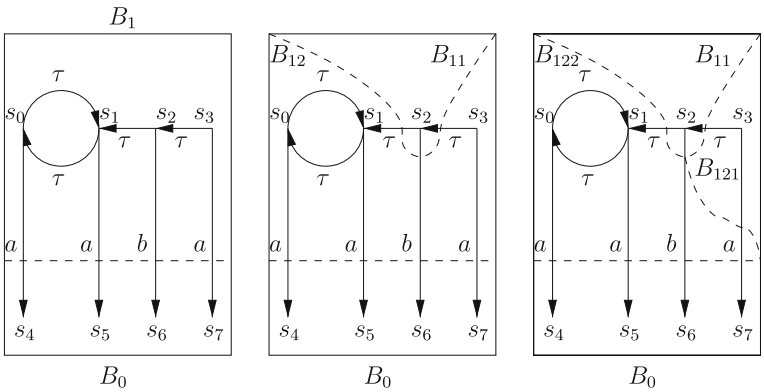

Fig. 6. An example for solving RCPSO

Therefore, we can find in $O(m)$ time a splitter of the current partition or find in $O(m)$ time that the current partition is stable. If a splitter is found, it takes $O(m+n)$ time to construct the new partition. Moreover, it is not hard to check that the space complexity of the algorithm above is $O(m+n)$. Thus we have the following theorem:

Theorem 2 *The RCPSO problem can be decided in $O(n(m+n))$ time, using $O(m+n)$ space.*

Example 4 Let $(S, \rightarrow)$ be the LTS given in Fig. 6.

$$S = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7\} \text{ and}$$
$$\rightarrow = \{s_0 \xrightarrow{\tau} s_1, s_0 \xrightarrow{a} s_4, s_1 \xrightarrow{\tau} s_0, s_1 \xrightarrow{a} s_5, s_2 \xrightarrow{\tau} s_1, s_2 \xrightarrow{b} s_6, s_3 \xrightarrow{\tau} s_2, s_3 \xrightarrow{a} s_7\}.$$

At the beginning, let $B_0 = \{s_4, s_5, s_6, s_7\}$, $B_1 = \{s_0, s_1, s_2, s_3\}$, and $P_0 = \{B_0, B_1\}$. We find the coarsest partition P_f of P_0 as follows. The block B_0 is a splitter of B_1 with respect to b. Thus, B_1 is split into $B_{11} = \{s_2\}$ and $B_{12} = \{s_0, s_1, s_3\}$ by (B_0, a). Let $P_1 = \{B_0, B_{11}, B_{12}\}$. Then it is easy to see that B_{11} is a splitter of B_{12} with respect to τ. We split B_{12} to $B_{121} = \{s_3\}$ and $B_{122} = \{s_0, s_1\}$. The refinement P_2 of P_1 is stable with respect to all the blocks, and therefore, $P_f = P_2$.

3.3. The RCPSO problem can be used to decide orthogonal bisimulation on finite LTS's

To decide orthogonal bisimulation of two states in a finite LTS, we can check whether they are in the same block of the coarsest stable partition P_f in the RCPSO problem with the initial partition P_0 consisting of two blocks: the first block contains all states that have no outgoing τ transitions and the second block contains the remaining states in this LTS. It takes $O(m + n)$ time to construct P_0, using $O(m + n)$ space.

Theorem 3 *Let $(S, \rightarrow)$ be a finite LTS, and let P_f be the final partition obtained after applying the RCPSO algorithm on an initial partition P_0 containing two blocks B_1 and B_2, where B_1 consists of all states in S that have no outgoing τ transitions and $B_2 = S \setminus B_1$. Then $\sim_{P_f} = \leftrightarrows_o$.*

Proof. This follows from the following two facts:

1. $\sim_{P_f} \subseteq \leftrightarrows_o$. It follows from Theorem 1 that P_f exists. We show that if $s \sim_{P_f} r$ then $s \leftrightarrows_o r$. By the definition of P_f, if $s \xrightarrow{a} s'$ then there exists r' such that $r \xrightarrow{a} r'$ and $s' \sim_{P_f} r'$. In the case that $s \xrightarrow{\tau} s'$, since P_f refines P_0, $r \xrightarrow{\tau} r'$. Moreover, there is an $n \geqslant 0$ and there are $r_0, \ldots, r_n \in S$ such that $r_0 = r$, for all $0 \leqslant i < n : s \sim_{P_f} r_i$ and $r_i \xrightarrow{\tau} r_{i+1}$, and $s' \sim_{P_f} r_n$. This implies that P_f is an orthogonal bisimulation equivalence.
2. $P_f \supseteq \leftrightarrows_o$. Orthogonal bisimulation equivalence $\leftrightarrows_o$ induces a stable partition that refines P_0 on S. As P_f is the coarsest stable partition that refines P_0, $\leftrightarrows_o \subseteq P_f$. □

The complexity for deciding orthogonal bisimulation is $O(n(m + n))$ time, using $O(m + n)$ space.

4. Concluding remarks

In this paper, we have presented an algorithm for deciding orthogonal bisimulation. Our algorithm is based on the well-known algorithm for deciding branching bisimulation given by Groote and Vaandrager in [GV90]. The difference between the two algorithms is that in our algorithm, transition systems may have cycles of silent steps. This makes the problem addressed in this paper more complicated. For instance, instead of dealing with states, we have to deal with sets of states called inert components. Nevertheless, we have shown that the complexity of our algorithm remains the same as that of [GV90]. Thus, it takes $O(n(m + n))$ time to decide orthogonal bisimilarity in finite state transition systems using $O(m + n)$ space. This thereby answers the open question in [BPZ03].

References

[AHU74] Aho AV, Hopcroft JE, Ullman JD (1974) The design and analysis of computer algorithms. Addison-Wesley, Reading

[BCG88] Browne MC, Clarke EM, Grumberg O (1988) Characterizing finite Kripke structure in propositional temporal logic. Theor Comput Sci 59(1–2):115–131

[BG87] Baeten JCM, van Glabbeek RJ (1987) Another look at abstraction in process algebra. In: Ottmann Th (ed) ICALP 87, Vol 267 of Lecture Notes in Computer Science, Springer, Heidelberg, pp 84–94

[BPZ03] Bergstra JA, Ponse A, van der Zwaag MB (2003) Branching time and orthogonal bisimulation equivalence. Theor Comput Sci 309:313–355

[Gla94] van Glabbeek RJ (1994) What is branching time semantics and why to use it? Bull EATCS 53:190–198

[GV90] Groote JF, Vaandrager FW (1990) An efficient algorithm for branching bisimulation and stuttering equivalence. In: Paterson MS (ed), ICALP 90, Vol 443 of Lecture Notes in Computer Science. Springer, Heidelberg, pp 626–638

[GW96] van Glabbeek RJ, Weijland WP (1996) Branching time and abstraction in bisimulation semantics. J ACM 43:555–600

[HM85] Hennessy M, Milner R (1985) Algebraic laws for nondeterminism and concurrency. J ACM 32:137–161

[Mil80] Milner R (1980) A calculus of communicating systems, Vol 92 of Lecture Notes in Computer Science. Springer, Heidelberg
[Mil81] Milner R (1981) A modal characterisation of observable machine behaviour. In: Astesiano G, Böhm C (eds), CAAP 81, Vol 112 of Lecture Notes in Computer Science. Springer, Heidelberg, pp 25–34
[PT87] Paige R, Tarjan R (1987) Three partition refinement algorithms. SIAM J Comput 16(6):973–989
[Vu05] Vu TD (2005) The compression structure of a process. Inf Process Lett 96:225–229

Received 4 July 2006
Revised 20 December 2006
Accepted 4 January 2007 by J. Parrow
Published online 10 March 2007