Information Processing Letters 44 (1992) 333–343
North-Holland

28 December 1992

Failure-equivalent transformation of transition systems to avoid internal actions *

Gang Luo, Gregor v. Bochmann, Anindya Das and Cheng Wu

Département d'Informatique et de Recherche Opérationelle, Université de Montréal, Montréal, Québec, Canada H3C 3J7

Communicated by F. Dehne **
Received 31 October 1991
Revised 24 July 1992

Abstract

Luo, G., G. v. Bochmann, A. Das and C. Wu, Failure-equivalent transformation of transition systems to avoid internal actions, Information Processing Letters 44 (1992) 333–343.

Labelled transition systems are often used as a theoretical framework for modelling communication protocols or distributed software systems. In order to model the deadlock properties of the systems, the internal transitions of the submodules are often represented globally by so-called internal actions. However, certain verification and testing methods assume that the specification of the system is given in the form of a transition system without internal actions. This paper shows that most labelled transition systems with internal actions can be transformed into a failure-equivalent labelled transition system without internal actions. A sufficient condition is given, and an algorithm is described which does the transformation for any transition system which satisfies the condition. This algorithm can be applied to use the above verification and testing methods also for specification including internal actions.

Keywords: Algorithms; software validation; process algebra; labelled transition system; failure equivalence; LOTOS

1. Introduction

Labelled transition systems (LTSs) are often taken as the basis for communication software specifications [1,3]. LOTOS [1], a communication software specification language, is based on LTSs. Due to their concise and well-defined mathematical formulation, many methods in the area of testing and verification are also based on LTSs [2,4,7] and LOTOS [9,10]. *Internal actions* (spontaneous transitions) in LTSs are very useful features for protocol specification and are often used to represent many protocol features abstractly [9], for instance to model the acknowledgement policies in a transport protocol. Internal actions also are used to model "time-out" features in communication protocols. The composition of several LTSs which communicate with one another when viewed as a single module also gives rise to internal actions. The usefulness of internal actions (*spontaneous transitions*) in representing many different features in an abstract and concise manner has led to its inclusion in the

Correspondence to: G. Luo, Département d'Informatique et de Recherche Opérationelle, Université de Montréal, C.P. 6128, Succ. A, Montréal, Québec, Canada H3C 3J7. Email: luo@iro.umontreal.ca.

* This work was supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols at the University of Montreal, Canada.

** *Editor's Note*: In the course of revisions requested by the Reviewers the length of this paper grew beyond the usual IPL limits. Any subsequent reduction would have impaired the presentation, thus I decided to publish the paper in extenso. This should not be construed as an indication of any cange of the editorial policy on the length of papers as specified in the "Instructions for Authors" (p. 3 of the covers).

next version of CCITT SDL [12]. However, internal actions in LTSs often give rise to situations which are difficult to deal with when verification and testing are considered. Therefore, some existing methods and tools do not consider internal actions. For example, the testing method given in [7] only considers finite LTSs without internal actions. CSP [8] also does not consider internal actions. An approach solving this problem is to develop transformation methods which can transform LTSs with internal actions to LTSs without internal actions which preserve certain equivalence relations. We consider in this paper only finite LTSs; and a finite representation of processes without value-passing expressed in some process algebra, e.g. CCS, is given by a systematic construction of the corresponding automata [13].

Many relations of LTSs are presented [2,3] to answer the question of what is conformance between specifications and their implementations. We study in this paper the transformation which preserves *failure equivalence* [2,5,6], an equivalence relation which is particularly interesting in testing [7]. We first show that there does not exist any algorithm which can transform all possible LTSs with internal actions to failure-equivalent LTSs without internal actions, but we find that an LTS can be transformed into failure-equivalent LTSs without internal actions if its initial state is *stable*. A state is said to be *stable* if no internal action starts from it. We also present a method to transform an LTS with internal actions into a failure-equivalent LTS without internal actions under the condition that the initial state is stable. We note that this condition is not a stringent condition in practice.

The transformation described in this paper can be used to extend the test selection method of [7]. [7] presents a method to generate test sequences from nondeterministic machine of specifications (which are modeled by LTSs) in order to test the implementations. The method assumes, however, that there are no internal actions in the nondeterministic specifications. Combined with our transformation approach, the method of [7] can be extended to test cases selection for nondeterministic finite state machines with internal actions, by first transforming the specification into an equivalent specification without internal actions. This transformation method may also be used in the area of protocol verification [11] where the result is often achieved for the LTSs without internal actions.

The rest of the paper is organized as follows. Section 2 introduces the basic definitions and notations used in this paper. Section 3 presents the transformation algorithm; the correctness of the algorithm is proved in the Appendix.

2. Basic definitions and notations

In this section, we introduce some basic definitions and notations. We first give the definition of LTSs [1,7,14].

Definition. *Labelled transition system (LTS).* An LTS is defined as a 4-tuple $\langle \mathrm{St}, L, T S_0 \rangle$, where

(1) St is a non-empty set of states,

(2) L is a set of observable actions,

(3) $T = \{ \rightarrow^{\mu} \mid \rightarrow^{\mu} : \mathrm{St} \times \mathrm{St} \text{ and } \mu \in L \cup \{i\} \}$ (i is an internal action),

(4) S_0 in St is the initial state of the system.

We write $P \rightarrow^{\mu} P'$ for a pair of states P and P' that belongs to the relation $\rightarrow^{\mu}$; it is also called a transition. $\rightarrow^{i}$ represents internal, non-observable transitions [1,2].

We use the notation shown in Table 1 for transitions and sets of observable and non-observable transitions that are relevant within a given LTS.

The following definitions of refusal function and failure equivalence are given in the form of [14]. For the sake of convenience, we also use LTS names to represent their initial states in the rest of the paper.

Definition. *Refusal function* of an LTS S. The refusal function of an LTS S, $R : L^* \rightarrow \mathit{powerset}(\mathit{powerset}(L))$, is defined for each σ in L^* by:

$$R(\sigma) = \{ A \mid A \subseteq L, \text{ and } \exists P \in \mathrm{St} \text{ such that } S \Rightarrow^{\sigma} P \text{ and } \forall a \in A,\ P \not\Rightarrow^{a} \}.$$

Table 1
Notation for labelled transition systems

notation	meaning
L	set of observable actions; $a, b, c, \ldots$ denote elements of L
L^*	set of strings over L; σ denotes such strings
L'	$L \cup \{i\}$; μ denotes elements of L'
St	set of states; $A, B, C, \ldots, I, P, Q$ and S denote such states
$P \rightarrow^{\mu_1 \cdots \mu_n} Q$	there exist P_k for $0 \leqslant k \leqslant n$ such that $P = P_0 \rightarrow^{\mu_1} P_1 \cdots \rightarrow^{\mu_n} P_n = Q$
$P \rightarrow^{\mu_1 \cdots \mu_n}$	there exists Q such that $P \rightarrow^{\mu_1 \cdots \mu_n} Q$
$P \not\rightarrow^{\mu_1 \cdots \mu_n}$	no Q exists such that $P \rightarrow^{\mu_1 \cdots \mu_n} Q$
$P \Rightarrow^{\epsilon} Q$	$P \rightarrow^{i^n} Q$ $(1 \leqslant n)$ or $P = Q$ (note: i^n means n times i)
$P \Rightarrow^{a} Q$	there exist P_1, P_2 such that $P \Rightarrow^{\epsilon} P_1 \rightarrow^{a} P_2 \Rightarrow^{\epsilon} Q$
$P \Rightarrow^{a_1 \cdots a_n} Q$	there exist P_k for $0 \leqslant k \leqslant n$ such that $P = P_0 \Rightarrow^{a_1} P_1 \cdots \Rightarrow^{a_n} P_n = Q$
$P \Rightarrow^{\sigma} Q$	$P \Rightarrow^{a_1 \cdots a_n} Q$ with $\sigma = a_1 \ldots a_n$
$P \Rightarrow^{\sigma}$	there exists Q such that $P \Rightarrow^{\sigma} Q$
$P \not\Rightarrow^{\sigma}$	no Q exists such that $P \Rightarrow^{\sigma} Q$

For specification-based testing, one should answer the question of what is the relation between a valid implementation F and its specification S. The relation is based on the following two intuitive notions: (1) everything that F does must be allowed by S and (2) everything prescribed by S should be implemented in F. The failure equivalence given below is a formalization of the notions [2,5,6].

Definition. *Failure equivalence* (or Testing Equivalence). The failure equivalence relation between two LTSs F and S holds iff for every σ in L^*, $R_F(\sigma) = R_S(\sigma)$.

In order to facilitate the presentation of our method, we require the following notations. An *LTS-Graph* is a labelled directed graph of an LTS where each node is labelled by a state name, each edge is labelled by an action name, and if $P \rightarrow^{\mu} Q$ then there is an edge from P to Q with label μ. Figure 1 shows an example of LTS-Graphs. An *i-Path* is a directed path with all of its edges labelled with i.

3. Algorithm of transformation

3.1. General idea of the transformation

In order to transform an LTS-Graph into a failure-equivalent LTS-Graph without internal actions we proceed in the following manner. We first eliminate all directed cycles of edges labelled i. This is done by repeatedly replacing the nodes in a cycle thus identified by a single node. All incoming and outgoing edges of the nodes in the cycle are now attached to the node replacing them. Once all directed cycles of edges labelled i have been eliminated, we proceed to the second phase.

In the second phase, we first remove redundant edges labelled i. An edge labelled i between nodes A and B is redundant if there exists another distinct path between A and B consisting only of edges labelled i. We then identify an i-Path and remove the last edge labelled i on this path, adding additional edges as follows. Let A and B be the head and the tail of the last edge on the i-Path. To each incoming edge of A we add an incoming edge to B and to each outgoing edge of B we add an outgoing edge to A. This last step is repeated until all edges labelled i are removed. The transformation procedures are given in Algorithms 1 and 3. Figure 1 shows a simple example to illustrate the idea of the transformation.

3.2. Impossibility of transformation in certain cases

It must be noted that there exist LTS-Graphs for which no failure-equivalent LTS-Graphs without internal actions can be found. An example of

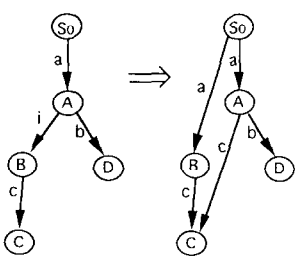

Fig. 1. An example to illustrate the transformation.

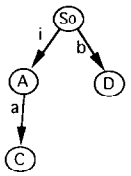

Fig. 2. An LTS ($L = \{a, b\}$) without any failure-equivalent LTS without i.

such an LTs-Graph which does not have an equivalent LTS-Graph without internal actions is shown in Fig. 2. We present the argument as follows.

Assume on the contrary that we have a failure-equivalent LTS-Graph F for the LTS-Graph S in Fig. 2. From the example in Fig. 2, we have

$$R_S(\varepsilon) = powerset(\{b\}).$$

Since both of $R_S(a)$ and $R_S(b)$ are not empty, neither $R_F(a)$ nor $R_F(b)$ should be not empty. Therefore we have $F \Rightarrow^a$ and $F \Rightarrow^b$. Furthermore, since there is not any i in F we have $F \rightarrow^a$ and $F \rightarrow^b$, which means

$$R_F(\varepsilon) = \{\ \{\ \}\ \}.$$

Therefore, $R_S(\varepsilon)$ is not equal to $R_F(\varepsilon)$, which is contrary to our assumption that S and F are failure-equivalent.

On the other hand, for a finite LTS, if the initial state S_0 is stable then our algorithm is guaranteed to find a failure-equivalent LTS without internal actions.

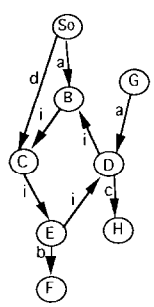

Fig. 3. An example to illustrate Algorithm 1.

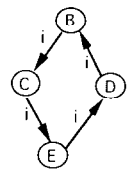

Fig. 4. An i-Cycle.

3.3. Algorithm

We consider only finite LTSs. Given an LTS-Graph, the following algorithm removes all directed cycles in which each edge is labelled i (i-Cycles).

Algorithm 1. i-Cycle elimination.
Input: LTS-Graph.
Output: LTS-Graph which does not contain any i-cycle.
Step 1: Find an i-Cycle in the LTS-Graph. Eliminate all edges of the i-Cycle, and collapse all nodes in the i-Cycle to form a single node.
Step 2: Repeat Step 1 until all i-Cycles are eliminated.
Step 3: Find an edge labelled i such that both the tail and head nodes have only one outgoing edge respectively. Eliminate the edge, and collapse the tail and head to form a single node.
Step 4: Repeat Step 3 until no progress can be made.
[End of algorithm].

As an example, we consider the LTS-Graph shown in Fig. 3 which contains an i-Cycle (Fig. 4). Figure 5 shows the failure-equivalent LTS without any i-Cycle. Steps 3 and 4 are used to reduce the complexity of the algorithm; the algorithm can work correctly even without Steps 3 and 4.

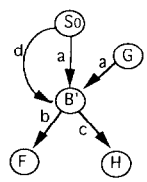

Fig. 5. The result obtained by applying Algorithm 1.

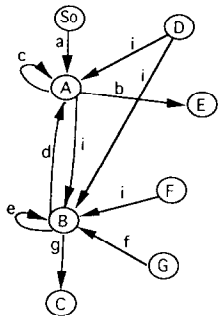

Fig. 6. An LTS with $L = \{a, b, c, d, e, f, g\}$.

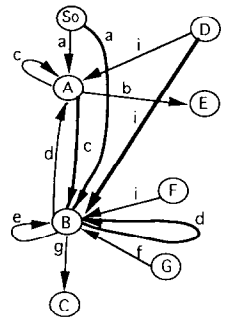

Fig. 8. The LTS of Fig. 6 transformed after Step 4 of Algorithm 3.

Theorem 2. *The LTS-Graph resulting from the application of Algorithm* 1 *is failure-equivalent to the original LTS-Graph. Algorithm* 1 *terminates after a finite number of steps.*

For the proof see the Appendix.

Algorithm 3. LTS transformation.

Input: LTS-Graph without i-Cycle.

Output: LTS-Graph without i.

Condition of applicability: The initial state S_0 is a stable state.

Step 1: Find an edge labelled i, say "e", for which there exists an i-Path having same starting node and ending node as "e" and not containing "e"; then delete the "e". Repeat this operation until no progress can be made (for example, according to Step 1, we obtain the result in Fig. 7 from Fig. 6 by deleting the edge from D to B).

Step 2: Find an edge with i, say "e", with tail A and head B, such that there is no outgoing edge labelled i from B.

Step 3: Delete the edge "e" found in Step 2.

Step 4: For every edge from some node P to A with label μ, add an edge labelled μ from P to B. (For example, we can obtain the LTS-Graph of Fig. 8 from the LTS-Graph of Fig. 7 by applying Steps 1, 2, 3 and 4.)

Step 5: If A does not have any outgoing edge labelled i after the edge e is deleted in Step 3, then for every edge from B to some node P with label μ, add an edge labelled μ from A to P. (Figure 9 shows the LTS-Graph obtained from the LTS-Graph in Fig. 8 by applying Step 5.)

Step 6: Repeat Steps 1–5 until there are no edges labelled i in the resulting LTS-Graph.

[End of algorithm].

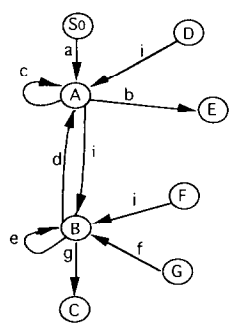

Fig. 7. The LTS of Fig. 6 transformed after Step 1 of Algorithm 3.

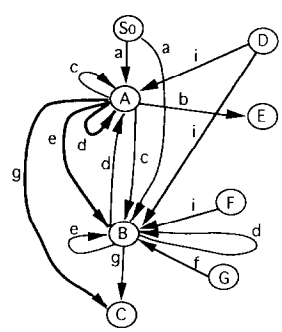

Fig. 9. The LTS of Fig. 6 transformed after Step 5 of Algorithm 3.

We make the following observations regarding Algorithm 3. Step 1 is used merely to reduce the complexity of the algorithm; the algorithm can work correctly even without Step 1. Step 4 ensures that if a σ in L^* can cause a deadlock before the transformation, then σ can also do so after the transformation. Step 5 ensures that if a σ in L^* cannot cause a deadlock before the transformation, the σ cannot also cause a deadlock after the transformation. If the A still has some outgoing edges labelled i after the edge e is deleted in Step 3, no edges will be added in Step 5.

We show that Step 4 and 5 do not produce any i-Cycle. Since all edges introduced in Step 4 end in B and there is no i-Path starting from B, Step 4 does not introduce any new i-Cycle. Since there is no i-Path starting from B, Step 5 does not introduce any new edge labelled i.

Theorem 4. *The LTS-Graph resulting from the application of Algorithm 3 is failure-equivalent to the original LTS-Graph. Algorithm 3 terminates after a finite number of steps.*

See the Appendix for a proof.

By applying Algorithm 1 and Algorithm 3, under the condition of applicability of Algorithm 3, we can transform a given LTS with internal actions to a failure-equivalent LTS without internal actions.

4. Conclusion

By using Algorithm 1 and Algorithm 3, we can transform all LTSs with internal actions, except for those having internal actions from their initial states, to the failure-equivalent LTSs without internal actions. In order to take advantage of this method, people should be encouraged to use LTSs without internal actions from the initial states. We consider in the paper the failure-equivalent transformations to avoid internal actions. Further work is needed for transformation algorithms to avoid internal actions concerning the other equivalence relations [3,14], such as bisimulation equivalence.

Appendix

In order to prove Theorems 2 and 4, we require the following definitions.

Definition. *S-after-σ.* S-after-σ is a set of states which can be reached by applying sequence σ starting from state S. Let $\sigma \in L^*$, $S \in \mathrm{St}$; we have

$$S\text{-after-}\sigma = \{P \mid P \in \mathrm{St} \text{ and } S \Rightarrow^{\sigma} P\}.$$

For the example in Fig. 9, S_0-after-$a = \{A, B\}$. We now give in the following a so-called *Node refusal function* of an LTS.

Definition. *Node refusal function* of an LTS S. The node refusal function of an LTS S, $Rn: \mathrm{St} \to powerset(powerset(L))$, is defined for each state Q in St by:

$$Rn(Q) = \{A \mid A \subseteq L, \exists P \in Q\text{-after-}\varepsilon \text{ such that } (\forall a \in A,\ P \not\Rightarrow^{a})\}.$$

Lemma A.1. (a) *If a given LTS S does not contain any i-Cycle, then*

$$Rn_S(Q) = \bigcup_{P \in Q\text{-after-}\varepsilon} Rn_S(P) = \bigcup_{\substack{P \in Q\text{-after-}\varepsilon \\ P \text{ is a stable state}}} Rn_S(P).$$

(b) *The refusal function of a given LTS S can be presented as follows:*

$$R_S(\sigma) = \bigcup_{P \in S\text{-after-}\sigma} Rn_S(P).$$

Proof. (Skeleton) (a) From the definition of the node refusal function, we have

$$\begin{aligned} Rn_S(Q) &= \{A \mid A \subseteq L, \exists P \in Q\text{-after-}\varepsilon \text{ such that } \exists P' \in P\text{-after-}\varepsilon \text{ and } (\forall a \in A,\ P' \not\Rightarrow^{a})\} \\ &= \{A \mid \exists P \in Q\text{-after-}\varepsilon \text{ such that } A \in Rn_S(P)\} \\ &= \bigcup_{P \subset Q\text{-after-}\varepsilon} Rn_S(P). \end{aligned}$$

Similarly, it can be proved that

$$Rn_S(Q) = \bigcup_{\substack{P \in Q\text{-after-}\varepsilon \\ P \text{ is a stable state}}} Rn_S(P).$$

(b) From the definitions of the refusal function and the node refusal function, we have

$$\begin{aligned} R_S(\sigma) &= \{A \mid A \subset L, \exists P \in S\text{-after-}\sigma \text{ such that} \\ &\qquad \exists P' \in P\text{-after-}\varepsilon \text{ and } (\forall a \in A,\ P' \not\Rightarrow^a)\} \\ &= \{A \mid \exists P \in S\text{-after-}\sigma \text{ such that } A \in Rn_S(P)\} \\ &= \bigcup_{P \in S\text{-after-}\sigma} Rn_S(P). \qquad \square \end{aligned}$$

Theorem 1. *The LTS-Graph resulting from the application of Algorithm* 1 *is failure-equivalent to the original LTS-Graph. Algorithm* 1 *terminates after a finite number of steps.*

Proof. (Skeleton) Let S be a given LTS-Graph and F the LTS-Graph after Step 1. Let W be the set of states in the i-Cycle identified in Step 1, B the node formed in Step 1. The following can be proved:

for a state P in S,

if $P \in W$, then $Rn_S(P) = Rn_F(B)$,

otherwise, $Rn_S(P) = Rn_F(P)$.

Then, by using Lemma A.1(b) it can be proved that $\forall \sigma \in L^*\ R_S(\sigma) = R_F(\sigma)$.

By using an approach similar to the above, it can be proved that Step 3 preserves the failure-equivalence relation. The termination of the algorithm is easy to prove since a finite number of edges labelled i are assumed, and that the Steps 1 and 3 decrease the edges labelled i. $\square$

The proof of Theorem 2 consists of two parts: the proof that the refusal function remains invariant for every σ in L^* after applying Algorithm 2, and the proof of termination of Algorithm 2. In order to prove Theorem 4, we require the following lemmas.

Lemma A.2. *Let the LTS-Graph after Step* 1 *be* S *and the LTS-Graph after Steps* 3, 4 *and* 5 *be* F. *After any single application of Step* 2, 3, 4 *and* 5, *we have*

$$\forall \sigma \in L^* \quad S\text{-after-}\sigma = F\text{-after-}\sigma.$$

Proof. We prove the lemma in the following two parts. We assume in the following that S_0 is the initial state of S and F.

Part I. We first show by induction on the length of a sequence σ in L^* that the following is true:

$$P \in S\text{-after-}\sigma \text{ implies } P \in F\text{-after-}\sigma. \qquad (1)$$

Induction base: Let σ be a sequence in L^* of length 0, i.e., $\sigma = \varepsilon$. According to the condition of applicability of Algorithm 3, the initial state of S is a stable state. Furthermore, it is easy to see that Algorithm 3 will not add any transition with an internal action to a stable state. Therefore, we have

$$S\text{-after-}\sigma = F\text{-after-}\sigma = \{S_0\}.$$

Thus (1) holds for all sequences σ in L^* of length 0.

Induction step: Now assume that (1) holds for all sequences σ in L^* of length at most k, $k \geqslant 0$. Let $\sigma = \sigma_1.a$ be a sequence in L^* of length $k+1$, and $P \in F\text{-after-}\sigma$.

If the edge e found in Step 2 does not lie on any one of the paths from S to P by applying σ, then (1) holds. Now suppose that the edge e found in Step 2 lies on one of the paths from S to P by applying σ. Let A and B be the tail and head respectively of edge e. Two cases occur if edge e is considered:

Case 1: The last occurrence of the edge e on this path can be found when σ_1 is applied. From $P \in S\text{-after-}\sigma_1.a$, there must be two states C and D along the path such that

$$S_0 \Rightarrow^{\sigma_1} C \rightarrow^a D \Rightarrow^{\varepsilon} P \quad \text{in } S. \qquad (2)$$

From the assumption of Case 1, we have

$$C \rightarrow^a D \Rightarrow^{\varepsilon} P \quad \text{in } F. \qquad (3)$$

According to the induction hypothesis and (2), we have

$$C \in F\text{-aftcr-}\sigma_1. \tag{4}$$

Then, from (3) and (4), it follows that $P \in F$-after-$\sigma_1.a$.

Case 2: The last occurrence of the edge e on this path cannot bc found when σ_1 is applied. We have one of the following situations in S:

$$S_0 \Rightarrow^{\sigma_1} A' \rightarrow^a A \rightarrow^i B = P, \tag{5}$$

$$S_0 \Rightarrow^{\sigma_1} A' \rightarrow^a B' \Rightarrow^{\varepsilon} C \rightarrow^i B = P. \tag{6}$$

According to the induction hypothesis, for both situations (5) and (6), wc havc

$$S_0 \Rightarrow^{\sigma_1} A' \quad \text{in } F. \tag{7}$$

If (5) occurs then by the elimination of the edge e in Step 3 and the addition of the edges in Step 4, we have

$$A' \rightarrow^a B = P \quad \text{in } F. \tag{8}$$

If (6) occurs, please notice that the edge e must not be in the portion of the path corresponding to $B' \Rightarrow^{\varepsilon} C$, since there is not any i-Cycle in S. Then by the elimination of the edge e in Step 3 and the addition of the edges in Step 4, we have

$$A' \rightarrow^a B' \Rightarrow^{\varepsilon} C \rightarrow^i B = P \quad \text{in } F. \tag{9}$$

From (7), (8) and (9), (1) holds for all sequences σ in L^* of length $k+1$.

Thus by induction (1) is true for all sequences σ in L^*.

Part II. We now show by induction on the length of a sequence σ in L^* that the following is true:

$$P \in F\text{-after-}\sigma \text{ implies } P \in S\text{-after-}\sigma. \tag{10}$$

Induction base: Let σ be a sequence in L^* of length 0, i.e., $\sigma = \varepsilon$. Based on the same arguments as given in the proof of "induction base" of Part I, we have that: (10) holds for all sequences σ in L^* of length 0.

Induction step: Now assume that (10) holds for all sequences σ in L^* of length at most k, $k \geqslant 0$. Let $\sigma = \sigma_1.a$ be a sequence in L^* of length $k+1$, and $P \in F$-after-σ.

If no edge introduced in Steps 4 or 5 lies on any path from F to P by applying σ, then (10) holds. Now suppose that an edge introduced in Steps 4 or 5 lies on one of the paths from F to P by applying σ. Let e_1 be the last occurrence of an edge introduced in Steps 4 or 5 which lies on this path. Let C and D be the tail and head respectively of edge e_1. Similar to the "induction step" of Part I, two cases occur if edge e_1 is considered:

Case 1: The edge e_1 on this path can be found when σ_1 is applied. Based on the arguments similar to Case 1 of the "induction step" of Part I, and from the induction hypothesis, it follows that P belongs to S-after-σ.

Case 2: Otherwise, we have one of the following situations in F:

$$S_0 \Rightarrow^{\sigma_1} C \rightarrow^a D = P, \tag{11}$$

$$S_0 \Rightarrow^{\sigma_1} A' \rightarrow^a B' \Rightarrow^{\varepsilon} C \rightarrow^i D = P. \tag{12}$$

Suppose (11) occurs. If the edge $C \rightarrow^a D$ was introduced in Step 4, based on the induction hypothesis, then the following path can be found in S:

$$S_0 \Rightarrow^{\sigma_1} C \rightarrow^a A \rightarrow^i D = P,$$

where $A \rightarrow^i D$ corresponds to the edge eliminated in Step 3 and $C \rightarrow^a A$ is an edge in S. If the edge $C \rightarrow^a D$ was introduced in Step 5, based on the induction hypothesis, then the following path can be found in S:

$$S_0 \Rightarrow^{\sigma_1} C \rightarrow^i B \rightarrow^a D = P,$$

where $C \rightarrow^i B$ corresponds to the edge eliminated in Step 3 and $B \rightarrow^a D$ is an edge in S. Thus for this case (10) holds.

Now suppose (12) occurs. We note that the edge $C \rightarrow^i D$ could only have been introduced in Step 4 since no edge labelled i is introduced in Step 5. Moreover, from (12), based on the induction hypothesis, a path $S_0 \Rightarrow^{\sigma_1} A' \Rightarrow^a B'$ can be found in S. Thus the following path can be found in S:

$$S_0 \Rightarrow^{\sigma_1} A' \rightarrow^a B' \Rightarrow^{\varepsilon} C \rightarrow^i A \rightarrow^i D = P,$$

where $A \rightarrow^i D$ corresponds to the edge eliminated in Step 3 and $C \rightarrow^i A$ is an edge in S. Hence for this case (10) holds.

By induction, (10) holds for all σ in L^*.

From (1) and (10), the lemma holds. □

Lemma A.3. *Let the LTS-Graph after Step* 1 *be S and the LTS-Graph after Steps* 3, 4 *and* 5 *be F. After any single application of Steps* 2, 3, 4 *and* 5, *for all* σ *in* L^*,

$$\bigcup_{P \in S\text{-after-}\sigma} Rn_S(P) = \bigcup_{Q \in S\text{-after-}\sigma} Rn_F(Q).$$

Proof. Let S-after-$\sigma = W_1 \cup W_2$, where

(1) σ is in L^*,

(2) the states A and B found in Step 2 are in S-after-σ,

(3) W_1 only contains all states which cannot be reached from A by i-Paths,

(4) W_2 only contains all states which can be reached from A by i-Paths, including A.

Therefore, $W_1 \cap W_2 = \emptyset$, and $A, B \in W_2$.

Part I. We first show that

$$\bigcup_{P \in W_1} Rn_S(P) = \bigcup_{Q \in W_1} Rn_F(Q). \tag{13}$$

Consider $P \in W_1$. If the P is not the tail of any incoming edge of A, then Steps 3, 4 and 5 do not change the outgoing edges of state P; therefore $Rn_S(P) = Rn_F(P)$. Otherwise, P is the tail of an incoming edge of A, Steps 3 and 5 do not change the outgoing edges of state P, and Step 4 only adds edges with labels which are the labels of existing outgoing edges of P; therefore $Rn_S(P) = Rn_F(P)$ still holds. Hence, (13) holds.

Part II. We now show that

$$\bigcup_{P \in W_2} Rn_S(P) = \bigcup_{Q \in W_2} Rn_F(Q). \tag{14}$$

From Lemma A.1(a), we have

$$\bigcup_{P \in W_2} Rn_S(P) = Rn_S(A). \tag{15}$$

Two cases occur if the outgoing edges of state A are considered.

Case 1: A has only one outgoing edge labelled i, which is deleted in Step 3; i.e., $W_2 = \{A, B\}$. In this case, from Lemma A.1(a), we have

$$Rn_S(A) = Rn_S(B). \tag{16}$$

From $W_2 = \{A, B\}$, we have

$$Rn_F(A) \cup Rn_F(B) = \bigcup_{P \in W_2} Rn_F(P). \tag{17}$$

We now argue

$$Rn_S(B) = Rn_F(B). \tag{18}$$

If the B is not the tail of any incoming edge of A, then Steps 3, 4 and 5 do not change the outgoing edges of state B; therefore (18) holds. Otherwise, B is the tail of an incoming edge of A, Steps 3 and 5 do not change the outgoing edges of state B, and Step 4 only adds the edges with labels which are the labels of existing outgoing edges of B; therefore (18) still holds.

Due to the edges added in Step 5, we have $Rn_F(B) \supset Rn_F(A)$. Thus

$$Rn_F(B) = Rn_F(B) \cup Rn_F(A). \tag{19}$$

From (15), (16), (17), (18) and (19), it follows that (14) holds for this case.

Case 2: A has more than one outgoing edges labelled i. In this case, no edges are added in Step 5. From Lemma A.1(a), we have

$$Rn_S(A) = Rn_S(B) \cup \left(\bigcup_{P \in W_2 - \{A, B\}} Rn_S(P) \right). \tag{20}$$

From Lemma A.1(a) we also have

$$\begin{aligned} &\bigcup_{P \in W_2} Rn_F(P) \\ &\quad = Rn_F(B) \cup Rn_F(A) \\ &\quad = Rn_F(B) \cup \left(\bigcup_{P \in W_2 - \{A, B\}} Rn_F(P) \right). \end{aligned} \tag{21}$$

Using arguments which are similar to those given for (18) in Case 1, we have

$$Rn_S(B) = Rn_F(B). \tag{22}$$

ıments which are similar to those given (13), we have

$$\bigcup_{P\in W_2-\{A,B\}} Rn_S(P) = \bigcup_{P\in W_2-\{A,B\}} Rn_F(P). \tag{23}$$

From (15), (20), (21), (22) and (23), it follows that (14) also holds for this case.

Thus (14) holds for all cases. From (13) and (14), the lemma follows. □

We define two terms in order to prove Lemma A.4. For LTSs, a *maximal i-path* terminating at a state Q is defined to be an i-Path such that it is not a portion of any other i-Path terminating at the Q. A function *lengths* : St → positive integers, with St being the state set, is defined as follows: For $Q \in$ St,

$$lengths(Q) = \sum_{\substack{p \text{ is a maximal } i\text{-Path} \\ \text{terminating at } Q}} \text{the length of } p.$$

Now we prove Lemma A.4.

Lemma A.4. *Algorithm* 3 *terminates after a finite number of steps.*

Proof. Assume $sum = \Sigma_{Q\in \mathrm{St}}\ lengths(Q)$. Since finite LTSs contain only a finite number of internal actions, and since no i-Cycles in the LTS according the condition of application of Algorithm 3, *sum* is a finite positive integer. It is easy to see that Steps 1 and 2 do not increase *sum*. We now argue that *sum* is decreased by each single application of Steps 3, 4 and 5. Let A and B be the tail and head respectively of the edge identified in Step 2. Please notice that a single application of Steps 3, 4 and 5 does not change the following:

$$\sum_{Q\in \mathrm{St}-\{B\}} lengths(Q).$$

However, a single application of Steps 3, 4, and 5 decreases $lengths(B)$. Therefore, *sum* is decreased. According to the definition of *sum, sum* will not be less than zero. Therefore, Algorithm 3 terminates after a finite number of steps. □

For the example illustrated in Fig. 7 we have, $lengths(B) = 2 + 1$ *and* $sum = lengths(B) + lengths(A) = (2 + 1) + 1 = 4$. After Steps 3, 4 and 5, we obtain the figure shown in Fig. 9 where $sum = lengths(B) + lengths(A) = (1 + 1) + 1 = 3$, decreased by one.

Theorem 4. *The LTS-Graph obtained by applying Algorithm* 3 *is failure-equivalent to the input LTS-Graph. Algorithm* 3 *terminates after a finite number of steps.*

Proof. We divide the proof of the above theorem into two parts.

Part I. To prove that the refusal function remains invariant for every σ in L^* after applying Algorithm 3. The LTS-Graph is modified only in Steps 1, 3, 4 and 5. Clearly Step 1 does not change the refusal function. Let the LTS-Graph after Step 1 be S and the LTS-Graph after Steps 3, 4 and 5 be F. From Lemmas A.1(b), A.2 and A.3 we have

$$\forall \sigma \in L^*,$$

$$R_S(\sigma) = \bigcup_{P\in S\text{-after-}\sigma} Rn_S(P) = \bigcup_{P\in S\text{-after-}\sigma} Rn_F(P)$$

$$= \bigcup_{P\in F\text{-after-}\sigma} Rn_F(P) = R_F(\sigma).$$

Therefore, it proves Part I since Steps 3, 4, and 5 of the algorithm are only used sequentially as a whole.

Part II. The proof of termination of Algorithm 3. Lemma A.4 proves Part II. □

Since Part I and Part II are independent of the order of choosing edges labelled i, the edge to be deleted in Step 3 can be chosen arbitrarily.

Acknowledgment

The authors would like to thank Professor Brinksma and Dr. Pierre de Saqui-Sannes for helpful discussions, and thank the anonymous referees for their very useful comments.

References

[1] T. Bolognesi and E. Brinksma, Introduction to the ISO specification language LOTOS, *Comput. Networks ISDN Systems* **14** (1987) 25–59.

[2] E. Brinksma, A theory for the derivation of tests, in: S. Aggarwal and K. Sabnani, eds., *IFIP Protocol Specification, Testing and Verification*, Vol. VIII (North-Holland, Amsterdam, 1988) 63–74.

[3] E. Brinksma, On the design of extended LOTOS, Ph.D. Dissertation, University of Twente, The Netherlands, 1988.

[4] E. Brinksma, R. Alderen, R. Langerak, J. van de Lagemaat and J. Tretmans, A formal approach to conformance testing, in: J. de Meer, ed., *Proc. 2nd Internat. Workshop on Protocol Testing Systems*, Berlin, Germany (1989) 311–325.

[5] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe, A theory of communicating sequential process, *J. ACM* **31** (1984) 560–599.

[6] R. De Nicola, Extensional equivalences for transition systems, *Acta Inform.* **24** (1987) 211–237.

[7] S. Fujiwara and G. v. Bochmann, Testing nondeterministic finite state machines with fault coverage, in: J. Kroon, R.J. Heijink and E. Brinksma, eds., *Proc. 4th Internat. Workshop on Protocol Testing Systems*, Leidschendam, The Netherlands (North-Holland, Amsterdam, 1992) 267–280.

[8] C.A.R. Hoare, Communicating sequential processes, *Comm. ACM* **21** (1978) 666–677.

[9] K. Naik and B. Sarikaya, Testing Communiction Protocols, *IEEE Software* (January 1992) 27–37.

[10] D.H. Pitt and D. Freestone, The derivation of conformance tests from LOTOS specifications, *IEEE Trans. Software Engrg.* **16** (12) (1990) 1337–1343.

[11] P. de Saqui-Sannes and J.P. Courtiat, From the simulation to the varification of Estelle specifications, in: S.T. Vuong, ed., *Proc. IFIP 2nd Internat. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols* (North-Holland, Amsterdam, 1990) 393–407.

[12] SDL Newsletter, December 1991.

[13] D. Taubner, *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*, Lecture Notes in Computer Science **369** (Springer, Berlin, 1989).

[14] J. Tretmans, Test case derivation from LOTOS specifications, in: S.T. Vuong, ed., *Proc. IFIP 2nd Internat. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols* (North-Holland, Amsterdam, 1990) 345–359.