

Merging Behavior Specifications*

Ferhat Khendek and Gregor v. Bochmann
Département d'informatique et de recherche opérationnelle
Université de Montréal
C. P. 6128, Succ. A, Montréal, Que H3C 3J7, Canada

Abstract

This paper describes a method for merging behavior specifications modeled by transition systems. Given two behavior specifications B_1 and B_2 , $\text{Merge}(B_1, B_2)$ defines a new behavior specification that extends B_1 and B_2 . Moreover, provided that a necessary and sufficient condition holds, $\text{Merge}(B_1, B_2)$ is a cyclic extension of B_1 and B_2 . In other words, $\text{Merge}(B_1, B_2)$ extends B_1 and B_2 , and any cyclic trace in B_1 or B_2 remains a cyclic in $\text{Merge}(B_1, B_2)$. Therefore, in the case of cyclic traces of B_1 or B_2 , $\text{Merge}(B_1, B_2)$ transforms into $\text{Merge}(B_1, B_2)$, and may exhibit, in a recursive manner, behaviors of B_1 and B_2 . If $\text{Merge}(B_1, B_2)$ is a cyclic extension of B_1 and B_2 , then $\text{Merge}(B_1, B_2)$ represents the least common cyclic extension of B_1 and B_2 . This approach is useful for the extension and integration of system specifications.

1 Introduction

Formal specifications play an important role in the development life cycle of systems. They capture the user requirements. They can be validated against such requirements and used as basis for the design of implementations and test suites. A formal specification represents the reference in each step of the development life cycle of the required system. The design and the verification of the specification of a system is a very complex task. Therefore, methodologies for the design of formal specifications become very important.

Systems may consist of many distinct functions. During the design and the validation of the specification, these functions may be taken into consideration simultaneously. The validation of such specification may be a very complex task. In order to facilitate the design and validation of the

* This research was supported by a grant from the Canadian Institute for Telecommunications Research under the NCE program of the Government of Canada and by an IBM research fellowship.

specification of a multiple-functions system, the divide-and-conquer approach may be very useful. In this case, a specification for each function is designed and analyzed separately. These specifications are then combined to form the required system specification. The combination of these functions specifications should preserve the semantic properties of every single function specification.

From another point of view, system specifications may be enriched by adding new behaviors required by the user, such as adding new functionality to a given system specification. Different system specifications may be integrated. In both cases, the semantic properties of the given system specifications and behaviors should be preserved. Preserving semantic properties may, for instance, mean that the combined specification exhibits at least the behavior of each single specification without introducing additional failures for these behaviors. This is captured by the formal relation between specifications, called extension, introduced in [Brin 86]. Informally, a behavior specification B_2 extends a behavior specification B_1 , if and only if, B_2 allows any sequence of actions that B_1 allows, and B_2 can only refuse what B_1 can refuse, after a given sequence of actions allowed by B_1 .

Given two behavior specifications B_1 and B_2 , we may combine them into a new behavior specification B , such that B extends B_1 and B extends B_2 . By definition of the extension relation, B may exhibit behaviors of B_1 (respectively B_2), without any new failure for these behaviors. However, B may exhibit behaviors of B_1 and behaviors of B_2 , in an exclusive manner. In other words, B may exhibit only behaviors of B_1 or only behaviors of B_2 , once the environment has chosen a behavior of B_1 or a behavior of B_2 , respectively.

A behavior specification B may contain certain sequences of actions that may be repeated recursively. Such sequences of actions start from the initial state of B and reach the initial state of B_1 . They are called cyclic sequences of actions. We assume that the completion of a cyclic sequence of actions in B corresponds to the completion of B . In other words, we assume that the initial state of B represents the "final" state for the sequences of actions (functionalities) in B . We are interested in combining two behavior specifications B_1 and B_2 into a new specification B , such that, in the case of cyclic sequences of actions of B_1 or B_2 , B may exhibit, without any new failure, behaviors in B_1 and behaviors in B_2 , in a recursive manner. In other words, B extends B_1 and B_2 , and after a cyclic sequence of actions of B_1 or B_2 , B transforms into B' , with B' extends B_1 and B_2 , and after a cyclic sequence of actions of B_1 or B_2 , B' transforms into B'' , with B'' extends B_1 and B_2 , and so on. This is possible, if B extends B_1 and B_2 , and any cyclic sequence of actions in B_1 or B_2

remains cyclic in B . Therefore, after a cyclic sequence of actions of B_1 or B_2 , B transforms into B , which extends B_1 and B_2 . This new relation between behaviors is called cyclic extension.

In this paper, we describe a formal approach for merging behavior specifications modeled by transition systems. Given two behavior specifications B_1 and B_2 , we define a new specification behavior, called $\text{Merge}(B_1, B_2)$, which extends B_1 and B_2 . Moreover, provided that a necessary and sufficient condition holds, $\text{Merge}(B_1, B_2)$ is the least common cyclic extension of B_1 and B_2 .

We consider two models of transition systems, the Acceptance Graphs (AGs), which are similar to the Acceptance Trees of Hennessy [Henn 85] and the Tgraphs in [Clea 93], and the Labelled Transition Systems (LTSs) [Kell 76]. The merging of behavior specifications is, first, defined in the AGs model, which is more tractable mathematically than the LTSs model. The merging of LTSs is based on the merging of AGs and relies on a correspondence between LTSs and AGs, which is introduced in this paper.

The remainder of this paper is structured as follows. The next section introduces the LTSs model, some related equivalence relations and preorders and the notions of least common extension and least common cyclic extension. Section 3 introduces the AGs model, the related equivalences and preorders, the notions of least common extension and least common cyclic extension for AGs, and the correspondence between AGs and LTSs. The merging of two AGs G_1 and G_2 , $\text{Merge}(G_1, G_2)$, is defined in Section 4. Main properties of Merge are listed and an example of application is also provided in Section 4. In Section 5, the merging of LTSs is defined, as well as its properties and an example of application. In Section 6, our approach is compared to the related ones. In Section 7, we conclude. The proofs of the propositions and the theorem stated in this paper are provided in the Appendix.

2 Labelled Transition Systems

2.1 Model

An LTS is a graph in which nodes represent states, and edges, also called transitions, represent state changes, labelled by actions occurring during the change of state. These actions may be observable or not.

Definition 2.1 [Kell 76]

An LTS S is a quadruple $\langle St, L, T, s_0 \rangle$, where

- St is a (countable) set of states
- L is a (countable) set of observable actions
- $T \subseteq St \times (L \cup \{\tau\}) \times St$ is a set of transitions, where a transition from a state s_i to state s_j by an action μ ($\mu \in L \cup \{\tau\}$) is denoted by $s_i \xrightarrow{\mu} s_j$. τ represents the internal, nonobservable action ($\tau \notin L$).
- s_0 is the initial state.

An LTS $S = \langle St, L, T, s_0 \rangle$ represents a process interacting, in a synchronous manner, with the environment by executing the actions in $L \cup \{\tau\}$ following the rules specified by T. More exactly S represents a set of processes. Each state s_i of S corresponds to a process P represented by the LTS $\langle St, L, T, s_i \rangle$. In the following, we use the terms process and state as synonyms. We also may refer to an LTS by its initial state. All the definitions on the states are extended to LTSs and processes. The term "interaction" refers to an observable action.

A finite LTS (FLTS for short) is an LTS in which St and L are finite. For the graphic representation of the FLTSs, the initial state will be circled. The notations in Table 1 are used for the LTSs.

$P \xrightarrow{\mu_1 \dots \mu_n} Q$	$\exists P_i (0 \leq i \leq n)$ such that $P = P_0 \xrightarrow{\mu_1} P_1 \dots P_{n-1} \xrightarrow{\mu_n} P_n = Q$
$P \xrightarrow{\mu_1 \dots \mu_n}$	$\exists Q$ such that $P \xrightarrow{\mu_1 \dots \mu_n} Q$
$P \not\xrightarrow{\mu_1 \dots \mu_n}$	not $(P \xrightarrow{\mu_1 \dots \mu_n})$
$P \xrightarrow{\epsilon} Q$	$P = Q$ or $\exists n \geq 1 P \xrightarrow{\tau^n} Q$
$P \xrightarrow{a} Q$	$\exists P_2$ such that $P \xrightarrow{\epsilon} P_1 \xrightarrow{a} P_2 \xrightarrow{\epsilon} Q$
$P \xrightarrow{a_1.a_2 \dots a_n} Q$	$\exists P_i (0 \leq i \leq n)$ such that $P = P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \dots a_n \Rightarrow P_n = Q$
$P \xrightarrow{\sigma} Q$	$\exists Q$ such that $P \xrightarrow{\sigma} Q$
$P \not\xrightarrow{\sigma}$	not $(P \xrightarrow{\sigma})$
$Tr(P)$	$\{\sigma \in L^* \mid P \xrightarrow{\sigma}\}$
$out(P)$	$\{a \in L \mid P \xrightarrow{a}\}$

Notations:
 $\mu, \mu_i \in L \cup \{\tau\}$; $a, a_i \in L$ represent states; ϵ represents the empty trace,
 $\sigma = a_1.a_2 \dots a_n$, where "." notes the concatenation of events or the sequence of events (traces).

Table 1. Notations for LTSs

For a given LTS $S = \langle St, L, T, s_0 \rangle$, a trace from a given state s_i , is a sequence of interactions that S can perform starting from state s_i . The traces that S can perform from its initial state represent the traces of S. s_i after σ ($= \{s_j \mid s_i \xrightarrow{\sigma} s_j\}$) denotes the set of all states reachable from s_i by sequence σ . $out(s_i, \sigma)$ ($= \{s_j \in (s_i \text{ after } \sigma) \mid s_i \xrightarrow{\sigma} s_j\}$) denotes the set of all possible interactions after σ , starting from state s_i . A trace of S is cyclic, if and only if the set of states reachable by this trace is equal to the set of states reachable by the empty trace from the initial state. An elementary cyclic trace is a cyclic trace that is not prefixed by a nonempty cyclic trace. Note that, any cyclic trace results from the concatenation of elementary cyclic traces.

Definition 2.2 (Cyclic Trace for LTSs)

Given an LTS $S = \langle St, L, T \rangle$, a trace σ is a cyclic trace in S , iff

$$(s_0 \text{ after } \sigma) = \{s_i \in St \text{ such that } s_0 \xrightarrow{\varepsilon} s_i\}.$$

Definition 2.3 (Elementary Cyclic Trace for LTSs)

Given an LTS $S = \langle St, L, T \rangle$, a trace σ is an elementary cyclic trace in S , iff

- (1) σ is a cyclic trace, and
- (2) $\sigma' \in L^*$ and σ' is a cyclic trace in S .

2.2 Equivalences and Preorders

Intuitively, different LTSs may describe the same "observable behavior". Different equivalences have been defined corresponding to different notions of "observable behavior" [DeNi 87]. In the case of trace equivalence, two systems are considered equivalent if the set of all possible sequences (traces) of interactions that they may produce are the same.

Finer equivalences are obtained if the refusal (blocking) properties of the systems, which are in general non-deterministic, are also taken into account. $P \text{ ref } A$ means that P refuses to perform any interaction in A ($P \text{ a} \Rightarrow, \forall a \in A$). In other words, P deadlocks with any interaction a in A . A is called a refusal for P . Note that if A is a refusal for P , then any subset of A is a refusal for P .

$\text{Ref}(P, \sigma) = \{X \mid \exists Q \in (P \text{ after } \sigma) \text{ such that } P \text{ ref } X\}$ denotes the refusal set of P after σ .

Note that if $\sigma \notin \text{Tr}(P)$, then $\text{Ref}(P, \sigma) = \emptyset$.

Two systems are testing equivalent, if in addition to trace equivalence, they have the same refusal (blocking) properties [Brin 86].

Definition 2.4 (Testing Equivalence for LTSs)

Let S_1 and S_2 be two LTSs, S_1 and S_2 are testing equivalent, $S_1 \text{ te } S_2$, iff

- (1) $\text{Tr}(S_1) = \text{Tr}(S_2)$, and
- (2) $\forall \sigma \in L^*, \text{Ref}(S_2, \sigma) = \text{Ref}(S_1, \sigma)$.

For instance, the LTSs S_1, S_2 and S_3 in Figure 1 can perform the same sequences (**a**, **a.b**, **a.b.c**, **a.b.d**) of interactions (**a**, **b**, **c** and **d**). They have the same set of traces, they are trace equivalent. Moreover, the LTSs S_1 and S_2 have the same refusal properties. Because of nondeterminism, S_1

and S2 may both refuse interaction **c** (respectively **d**) after the sequence of interactions **a.b**. S1 and S2 are not distinguishable by external experiences. They are testing equivalent. However, S3 is not testing equivalent to S1 (and S2). S3 always accept interaction **c** or **d**, after the sequence **a.b**.

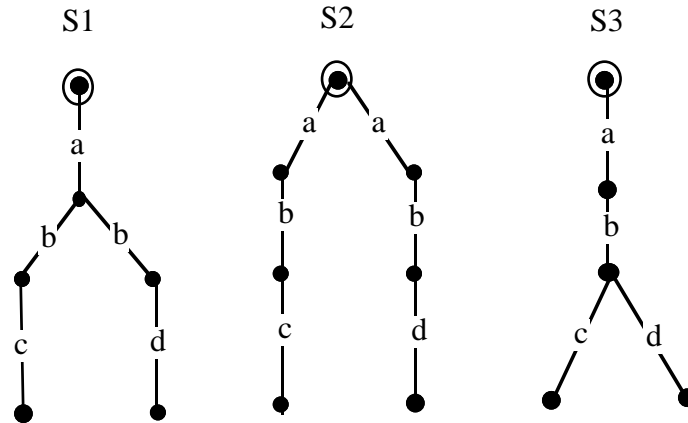


Figure 1. Examples for behavior equivalences.

Instead of considering the sets of interactions that may be refused, we may consider the sets of interactions that may be accepted. The notion of acceptance sets is dual to the notion of refusal sets. If $\text{Ref}(P, \sigma)$ is a refusal set, then the corresponding acceptance set $\text{Acc}(P, \sigma)$, is defined as the complement of the refusals in $\text{Ref}(P, \sigma)$ with respect to $\text{out}(P, \sigma)$.

$$\begin{aligned} \text{Acc}(P, \sigma) &= \{\text{out}(P, \sigma) - X \mid X \in \text{Ref}(P, \sigma)\} \\ &= \{X \mid \exists Q \in (P \text{ after } \sigma) \text{ such that } \text{out}(Q) = X \cup \text{out}(P, \sigma)\}. \end{aligned}$$

The following properties of $\text{Acc}(P, \sigma)$ can be derived from its definition:

- $\text{Acc}(P, \sigma) = \emptyset$ iff $\sigma \notin \text{Tr}(P)$,
- $\forall A1, A2 \in \text{Acc}(P, \sigma), A1 \cap A2 \in \text{Acc}(P, \sigma)$,
- $\forall A1, A2 \in \text{Acc}(P, \sigma)$, if $A1 \cup A3 = A2$, then $A3 \in \text{Acc}(P, \sigma)$.

Intuitively, a set of interactions X belongs to $\text{Acc}(P, \sigma)$, if and only if there is a state Q reachable from P by σ and X includes the set of interactions enabled in this state, but X is included in the set of all possible interactions of $(P \text{ after } \sigma)$. This definition corresponds to the acceptance sets definition in [Henn 85].

Condition (2) in Definition 2.4 may be stated in terms of acceptance sets as follows:

$$\forall \sigma \in L^*, \text{Acc}(S2, \sigma) = \text{Acc}(S1, \sigma).$$

Similar testing equivalence relations are defined in [Broo 85, DeNi 84, Henn 88]. They differ from the testing equivalence we consider in this paper, in the way the divergence (possibility of infinite sequence of internal actions) is dealt with.

Finer equivalences, the bisimulation equivalence (strong bisimulation, \approx) [Park 81] and the observation equivalence (weak bisimulation, \approx) [Miln 89], may be defined if the internal states of the two systems are taken into account. These relations are based on the notions of strong bisimulation [Park 81] and weak bisimulation [Miln 89], respectively.

Definition 2.1 (Strong Bisimulation)

A relation R is a strong bisimulation, if $(s_i, s_j) \in R$ implies that $\forall a \in (L \setminus \{\tau\})$, if $s_i \xrightarrow{a} s_k$ and $(s_k, s_l) \in R$, then $s_j \xrightarrow{a} s_l$ and $(s_k, s_l) \in R$.

Definition 2.2 (Weak Bisimulation)

A relation R is a weak bisimulation, if $(s_i, s_j) \in R$ implies that $\forall a \in (L \setminus \{\varepsilon\})$, if $s_i \xrightarrow{a} s_k$ and $(s_k, s_l) \in R$, then $s_j \xrightarrow{a} s_l$ and $(s_k, s_l) \in R$.

Two LTSs S_1 and S_2 , with s_{1_0} and s_{2_0} as initial state, respectively, are (strongly) bisimulation equivalent, $S_1 \approx S_2$, (respectively observation equivalent, $S_1 \approx S_2$), if and only if there is a strong bisimulation R (respectively weak bisimulation R) with $(s_{1_0}, s_{2_0}) \in R$. The observation equivalence of Milner is stronger than the testing equivalence, but weaker than the bisimulation equivalence. Two LTSs S_1 and S_2 , with s_{1_0} and s_{2_0} as initial state, respectively, are isomorphic, if and only if there is a strong bisimulation R , such that $(s_{1_0}, s_{2_0}) \in R$ and each state of S_1 is related to one and only one state of S_2 and vice et versa.

In addition to the equivalences, many preorders (reflexive and transitive relations) have been defined in the literature [DeNi 87, Henn 85, Brin 86]. The extension preorder defined in [Brin 86] is most appropriate for extending specification behaviors. Informally, S_2 extends S_1 , $S_2 \text{ ext } S_1$, if and only if S_2 may perform any sequence of interactions that S_1 may perform, and S_2 can not refuse what S_1 can not refuse after a given sequence of interactions allowed by S_1 [Brin 86]. The extension preorder induces the testing equivalence [Brin 86]. In other words, two specifications are testing equivalent if and only if each is the extension of the other. In the following, for a given set X , $P(X)$ denotes the power set of X , i.e. the set of subsets of X .

Definition 2.7

Let $A, B \subseteq P(L)$. $A \sqsubseteq B$, iff $\forall A_1 \in A, \exists B_1 \in B$ such that $B_1 \sqsubseteq A_1$.

The following definition of the extension introduced in [Ledu 90] is equivalent to the original one:

Definition 2.8 (Extension for LTSs)

Let S_1 and S_2 be two LTSs, $S_2 \text{ ext } S_1$, iff

- (1) $\text{Tr}(S_1) \subseteq \text{Tr}(S_2)$, and
- (2) $\forall \sigma \in \text{Tr}(S_1), \text{Acc}(S_2, \sigma) \subseteq \text{Acc}(S_1, \sigma)$.

For instance, the LTSs S_6 and S_7 in Figure 2 extend both of the LTSs S_4 and S_5 . S_6 (and S_7) may perform any sequence of interactions that S_4 (respectively S_5) may perform and S_6 can not refuse what S_4 (respectively S_5) may not refuse after a sequence of interactions allowed by S_4 (respectively S_5). However, S_8 does neither extend S_3 nor S_4 . Indeed, S_8 may perform any sequence of interactions that S_4 (respectively S_5) may perform, but S_8 may, for instance, refuse interaction b (respectively c) after sequence a , whereas S_4 (respectively S_5) never refuses to interaction b (respectively c) after sequence a .

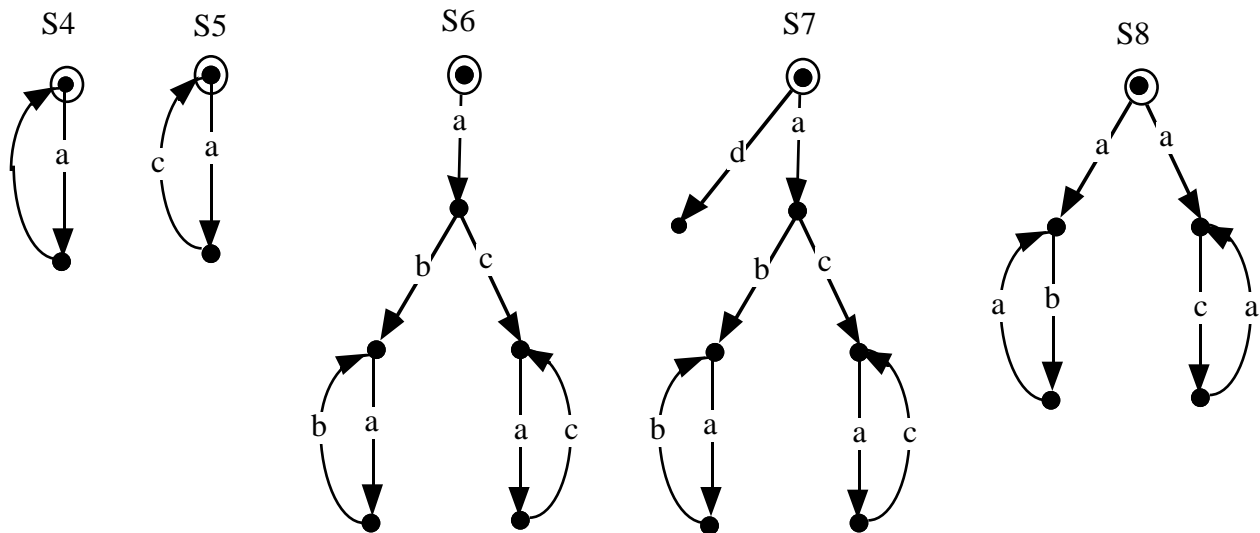


Figure 2. Extension of behaviors.

Among the common extensions of S_4 and S_5 , S_6 is the least one. In other words, any common extension of S_4 and S_5 is an extension of S_6 . For instance, S_7 extends S_6 . The least common extension is unique up to testing equivalence.

Definition 2.9 (Least Common Extension for LTSs)

Given three LTSs S_1 , S_2 and S_3 , such that $S_3 \text{ ext } S_1$ and $S_3 \text{ ext } S_2$,
 S_3 is the least common extension of S_1 and S_2 , iff
any common extension of S_1 and S_2 is also an extension of S_3 .

As introduced previously, in this paper we assume that the completion of a cyclic sequence of interactions in a given specification S corresponds to the completion of S . For instance, after performing $\mathbf{a.b}$, S_4 has completed its functionality and may repeat it in a recursive manner. The LTS S_6 , in Figure 2, extends both S_4 and S_5 . However, S_6 may exhibit only behavior $\mathbf{a.b}$ of S_4 in a recursive manner or only behavior $\mathbf{a.c}$ of S_5 in a recursive manner. S_6 does not exhibit behaviors of S_4 and behaviors of S_5 , in a recursive manner, contrarily to the LTS S_9 in Figure 3. Indeed S_9 extends both S_4 and S_5 and after performing a cyclic sequence of interactions in S_4 (respectively S_5) S_9 transforms into S_9 and offers again behaviors of S_4 and S_5 . S_9 may exhibit the behaviors $\mathbf{a.b.a.b\dots}$, $\mathbf{a.c.a.c\dots}$, $\mathbf{a.b.a.c.a.b.a.c, \dots}$ etc. A condition for S_9 to transform into S_9 after any cyclic trace of S_4 or S_5 , is that any cyclic trace in S_4 (respectively S_5) is a cyclic trace in S_9 . In this case, S_9 is called a cyclic extension of S_4 (respectively S_5).

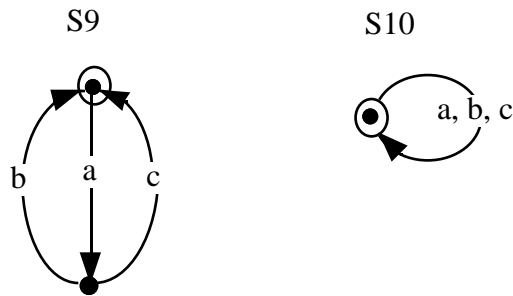


Figure 3. Cyclic extension of behaviors.

Definition 2.10 (Cyclic Extension for LTSs)

Let S_1 and S_2 be two LTSs. S_2 is a cyclic extension of S_1 , $S_2 \text{ extc } S_1$, iff

- (1) $S_2 \text{ ext } S_1$, and
- (2) any cyclic trace in S_1 is a cyclic trace in S_2 .

Since any cyclic trace results from the concatenation of elementary cyclic traces, any cyclic trace in S_1 is a cyclic trace in S_2 , if and only if any elementary cyclic trace in S_1 is a cyclic trace in S_2 . Among the common cyclic extensions of S_4 and S_5 shown in Figure 2, S_9 shown in Figure 3 is the least one. In other words, any common cyclic extension of S_4 and S_5 is a cyclic extension of S_9 . For instance, S_{10} , a cyclic extension of S_4 and S_5 , is also a cyclic extension of S_9 . Note that the least common cyclic extension of S_4 and S_5 , S_9 , extends the least common extension of S_4 and S_5 , S_6 .

Definition 2.11 (Least Common Cyclic Extension for LTSs)

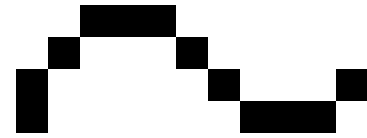
Given three LTSs S_1, S_2 and S_3 , such that $S_3 \text{ extc } S_1$ and $S_3 \text{ extc } S_2$, S_3 is the least common cyclic extension of S_1 and S_2 , iff any common cyclic extension of S_1 and S_2 is also a cyclic extension of S_3 .

The testing equivalence is refined into the cyclic testing equivalence, if the preservation of the cyclic traces is taken into account. Note that the cyclic extension is a preorder and it induces the cyclic testing equivalence.

Definition 2.12 (Cyclic Testing Equivalence for LTSs)

Let S_1 and S_2 be two LTSs. S_2 and S_1 are cyclic testing equivalent, $S_1 \text{ tec } S_2$, iff

- (1) $S_1 \text{ te } S_2$, and
- (2) any cyclic trace in S_1 is a cyclic trace in S_2 and reciprocally.



S_1 and S_2 have the same set of cyclic traces, as stated by condition (2) in Definition 2.12, if and only if S_1 and S_2 have the same set of elementary cyclic traces, since the concatenation of elementary cyclic traces leads a cyclic trace. Similarly to the testing equivalence, the strong bisimulation and the observation equivalence are also refined into the cyclic strong bisimulation (\approx_c) and the cyclic observation equivalence (\approx_c), respectively, when the preservation of the cyclic traces is taken into consideration.

3 Acceptance Graphs

3.1 Model

An AG is a bilabelled graph-structure. An AG is a graph in which nodes represent states, and transitions represent interactions occurring during state changes. Instead of modeling the nondeterminism by the labels of the transitions, the AGs model allows to keep such information in the labels of the states. Each state is labelled by a set of sets of interactions, called acceptance set, that the system may accept (perform) at this state. The outgoing transitions, from a given state, have distinct labels.

Definition 3.1 (Acceptance Graph)

An AG G is 5-tuple $\langle S_g, L, Ac, T_g, g_o \rangle$, where

- S_g is a (countable) non empty set of states.
- L is a (countable) set of interactions.

- $Ac: Sg \rightarrow P(P(L))$ is a mapping from Sg to a set of subsets of L .
 $Ac(g_i)$ is called the acceptance set of state g_i .
- $Tg: Sg \times L \rightarrow Sg$ is a transition function, where a transition from state g_i to state g_j by an interaction a ($a \in L$) is denoted by $g_i - a \rightarrow g_j$.
- g_0 is the initial state.

The AGs used in this paper are similar to the Acceptance Trees of Hennessy [Henn 85] and Agraphs in [Clea 93]. However, in our case, we do not distinguish between "closed" and "open" states, since divergence is not considered explicitly as in [Henn 85] or [Clea 93]. In this paper, any state g_i is labelled by an acceptance set, $Ac(g_i)$, which may be infinite or contain some infinite elements in the case where g_i is infinitely branching ($\{g_j \mid g_i - a \rightarrow g_j \text{ for some } a \in L\}$ is infinite). The mapping Ac and the transition function Tg should satisfy the following consistency constraints, which are similar to the consistency constraints defined for the "closed" states in [Henn 85]:

- $C_0: \forall g_i \in Sg, Ac(g_i) \neq \emptyset$.
- $C_1: \forall g_i \in Sg, A \in Ac(g_i)$ and $a \in A$, there is one and only one $g_j \in Sg$ such that $g_i - a \rightarrow g_j$.
- $C_2: \forall g_i \in Sg$, if $\exists g_j \in Sg$, such that $g_i - a \rightarrow g_j$, then $\exists A \in Ac(g_i)$ with $a \in A$.
- $C_3: \forall g_i \in Sg$, if $A_1, A_2 \in Ac(g_i)$, then $A_1 \cap A_2 \in Ac(g_i)$.
- $C_4: \forall g_i \in Sg$, if $A_1, A_2 \in Ac(g_i)$ and $A_1 \cap A_2 = A_3$, then $A_3 \in Ac(g_i)$.

A finite AG (FAG for short) is an AG in which Sg and L are finite. As for the LTSs, the initial state will be circled for the graphic representation of an FAG. The notations introduced in Table 1 will be used for the AGs with the same meaning as for the LTSs, since leaving the mapping Ac out of account, an AG can be seen as an LTS. In the case of AGs, the notation " g_i after σ " will denote the state g_j such that $g_i - \sigma \Rightarrow g_j$, instead of set of states in the case of LTSs. The notion of cyclic trace for AGs corresponds to that of cyclic path in the graph theory. A cyclic trace is a trace, of the initial state, that reaches the initial state. Similarly to the LTSs, an elementary cyclic trace, is a cyclic trace, which does not result from the concatenation of cyclic subtraces. Any cyclic trace results from the concatenation of elementary cyclic traces.

Definition 3.2 (Cyclic Trace for AGs)

Given an AG $G = \langle Sg, L, Ac, Tg, g_0 \rangle$, a trace σ is a cyclic trace in G iff $g_0 - \sigma \Rightarrow g_0$.

Definition 3.3 (Elementary Cyclic Trace for AGs)

Given an AG $G = \langle Sg, L, Ac, Tg, g_0 \rangle$, a trace σ is an elementary cyclic trace in G , iff

- (1) σ is a cyclic trace, and
- (2) $\sigma'(\sigma) = \sigma$ and σ' is cyclic trace in G .

An AG G may contain certain states that are not reachable (A state g_i is reachable iff $\exists \sigma \in \text{Tr}(G)$ such that $g_0 = \sigma \Rightarrow g_i$). The graph defined by the set of reachable states, their acceptance sets and their transitions as defined in G , denoted by $\text{reachable}(G)$, is an AG. It is obvious that $\text{reachable}(G)$ satisfies all the consistency requirements listed above.

Definition 3.4 (Reachable Part of an AG)

Given an AG $G = \langle Sg, L, Ac, Tg, g_0 \rangle$, the reachable part of G , $\text{reachable}(G)$, is an AG $G' = \langle Sg', L, Ac', Tg', g_0 \rangle$, where

- $Sg' = \{g_i \in Sg \mid \exists \sigma \in \text{Tr}(G) \text{ such } g_0 = \sigma \Rightarrow g_i\}$
- $\forall g_i \in Sg', Ac'(g_i) = Ac(g_i)$,
- $\forall g_i, g_j \in Sg', g_i \xrightarrow{a} g_j \in Tg' \text{ iff } g_i \xrightarrow{a} g_j \in Tg$.

3.2 Equivalences and preorders

Similarly to the LTSs, in the case of trace equivalence, two AGs G_1 and G_2 are considered equivalent, if and only if $\text{Tr}(G_1) = \text{Tr}(G_2)$. However, in the case of AGs, the testing equivalence and the observation equivalence coincide with the bisimulation equivalence. The LTS's structure is finer than the AG's structure. In this paper, we define the bisimulation for AGs as an instantiation of the Π -bisimulation introduced in [Clea 93].

Definition 3.5 (Bisimulation)

A relation $R \subseteq Sg \times Sg$ is a bisimulation, if $(g_i, g_j) \in R$ implies that

- $Ac(g_i) = Ac(g_j)$, $a \in L$,
- if $g_i \xrightarrow{a} g_k$ then $g_j \xrightarrow{a} g_l$ and $(g_k, g_l) \in R$,
- if $g_j \xrightarrow{a} g_l$ then $g_i \xrightarrow{a} g_k$ and $(g_k, g_l) \in R$.

Definition 3.6

Two AGs $G_1 = \langle Sg_1, L_1, Ac_1, Tg_1, g_{1_0} \rangle$ and $G_2 = \langle Sg_2, L_2, Ac_2, Tg_2, g_{2_0} \rangle$ are bisimulation equivalent, $G_1 \sim G_2$, if and only if there is a bisimulation R such that $(g_{1_0}, g_{2_0}) \in R$.

An alternative definition of the bisimulation equivalence for AGs is given by Proposition 3.1.

Proposition 3.1

Given two AGs $G_i = \langle Sg_i, L_i, Ac_i, Tg_i, g_{i_0} \rangle$, $i = 1, 2$; $G_1 \sim G_2$ iff $\text{Tr}(G_1) = \text{Tr}(G_2)$ and $(\forall \sigma \in \text{Tr}(G_1), Ac_1(g_{1_0} \text{ after } \sigma) = Ac_2(g_{2_0} \text{ after } \sigma))$.

Two AGs G_1 and G_2 , with g_{1_0} and g_{2_0} as initial state, respectively, are isomorphic, $G_1 =_g G_2$, if and only if there is a bisimulation R , such that $(g_{1_0}, g_{2_0}) \in R$ and each state of G_1 is related to one and only one state of G_2 and vice et versa.

Similarly to the LTSs, the extension relation is defined as follows:

Definition 3.7 (Extension for AGs)

Let G_1 and G_2 be two AGs. G_2 extends G_1 , $G_2 \text{ ext}_g G_1$, iff

- (1) $\text{Tr}(G_1) = \text{Tr}(G_2)$, and
- (2) $\forall \sigma \in \text{Tr}(G_1), \text{Ac}_2(g_{2_0} \text{ after } \sigma) = \text{Ac}_1(g_{1_0} \text{ after } \sigma)$.

In the case of AGs, the extension is a preorder that induces the bisimulation equivalence. From Proposition 3.1 and Definition 3.6, it is obvious that if $G_2 \text{ ext}_g G_1$ and $G_1 \text{ ext}_g G_2$, then $G_1 =_g G_2$. If we take into consideration the preservation of the cyclic traces, the extension and the bisimulation equivalence are refined into the cyclic extension and the cyclic bisimulation equivalence. Note that the cyclic extension preorder induces the cyclic bisimulation equivalence. Similarly to the LTSs, the cyclic traces of a given AG are preserved, if and only if its elementary cyclic traces are preserved, at least, as cyclic traces. Two AGs have the same set of cyclic traces, if and only if they have the same set of elementary cyclic traces.

Definition 3.8 (Cyclic Extension for AGs)

Let G_1 and G_2 be two AGs,

G_2 is a cyclic extension of G_1 , written $G_2 \text{ ext}_g G_1$, iff

- (1) $G_2 \text{ ext}_g G_1$,
- (2) any cyclic trace in G_1 is a cyclic trace in G_2 .

Definition 3.9 (Cyclic Bisimulation for AGs)

Let G_1 and G_2 be two AGs,

G_2 and G_1 are cyclic bisimulation equivalent, written $G_1 =_{c_g} G_2$, iff

- (1) $G_1 =_g G_2$, and
- (2) any cyclic trace in G_1 is a cyclic trace in G_2 and reciprocally.

The notions of least common extension and least common cyclic extension for AGs are defined in a similar way as for LTSs.

Definition 3.10 (Least Common Extension)

Given three AGs G_1 , G_2 and G_3 , such that $G_3 \text{ ext}_g G_1$ and $G_3 \text{ ext}_g G_2$, G_3 is the least common extension of G_1 and G_2 , iff any common extension of G_1 and G_2 is also an extension of G_3 .

Definition 3.11 (Least Common Cyclic Extension)

Given three AGs G_1 , G_2 and G_3 , such that $G_3 \text{ extc}_g G_1$ and $G_3 \text{ extc}_g G_2$, G_3 is the least common cyclic extension of G_1 and G_2 , iff any common cyclic extension of G_1 and G_2 is also a cyclic extension of G_3 .

3.3 Correspondence and transformations between AGs and LTSs

This section aims to define a correspondence between the LTSs and the AGs as well as the constructions for generating AGs from arbitrary LTSs and vice et versa. The correspondence between LTSs and AGs is based on the preservation of the traces, the acceptance sets and the cyclic traces.

Definition 3.12 (Correspondence between LTSs and AGs)

Given an LTS $S = \langle St, L, T, s_0 \rangle$ and an AG $G = \langle Sg, L, Ac, Tg, g_0 \rangle$, we say that G is the AG corresponding to S , $G = \text{ag}(S)$, iff

- (1) $\text{Tr}(S) = \text{Tr}(G)$,
- (2) $\forall \sigma \in \text{Tr}(G), \text{Ac}(g_0 \text{ after } \sigma) = \text{Acc}(s_0, \sigma)$,
- (3) any cyclic trace in S is a cyclic trace in G , and
- (4) any cyclic trace in G is a cyclic trace in S .

Note that, for a given LTS, the corresponding AG is unique up to the cyclic bisimulation equivalence. However, An AG may correspond to more than one LTS. These LTSs are cyclic testing equivalent. The following proposition is straightforward.

Proposition 3.2

Given two LTSs S_1, S_2 , and two AGs G_1, G_2 , such that $G_1 = \text{ag}(S_1)$ and $G_2 = \text{ag}(S_2)$, the following holds:

- (1) $S_2 \text{ ext } S_1$ iff $G_2 \text{ ext}_g G_1$.
- (2) any cyclic trace in S_1 is a cyclic trace in S_2 iff any cyclic trace in G_1 is a cyclic trace in G_2 .

Lemma 3.1 follows from Proposition 3.2, since the extension (respectively, the cyclic extension) induces the testing equivalence (respectively, the cyclic testing equivalence) in the case of LTSs and the bisimulation equivalence (respectively, the cyclic bisimulation equivalence) in the case of AGs.

Lemma 3.1

Given two LTSs S_1, S_2 , and two AGs G_1, G_2 , such that $G_1 = ag(S_1)$ and $G_2 = ag(S_2)$, the following holds:

- (1) $S_1 \text{ te } S_2 \text{ iff } G_1 \text{ }_g G_2$,
- (2) $S_1 \text{ tec } S_2 \text{ iff } G_1 \text{ }_{c_g} G_2$.

Lemma 3.2 follows from Proposition 3.2 and the definitions of least common extension and least common cyclic extension for LTS and AGs, respectively.

Lemma 3.2

Given three LTSs S_1, S_2, S_3 and three AGs G_1, G_2, G_3 , such that $G_1 = ag(S_1)$, $G_2 = ag(S_2)$ and $G_3 = ag(S_3)$, the following holds:

- (1) S_3 is the least common extension of S_1 and S_2 , iff G_3 is the least common extension of G_1 and G_2 .
- (2) S_3 is the least common cyclic extension of S_1 and S_2 , iff G_3 is the least common cyclic extension of G_1 and G_2 .

In the following proposition we define for an arbitrary LTS the corresponding AG. The definition of the corresponding AG for an arbitrary LTS is similar to the construction of a Tgraph from an arbitrary LTS in [Clea 93].

Definition 3.13 (ϵ -closure of a set of states) [Clea 93]

Given an LTS $S = \langle St, L, T, s_0 \rangle$, the ϵ -closure of a set of states $Qt \in P(St)$ (written Qt^ϵ), is defined as follows: $Qt^\epsilon = \{s_j \in St \mid \exists s_i \in Qt \text{ such that } s_i \xrightarrow{\epsilon} s_j\}$.

Proposition 3.3 (Definition of the AG corresponding to an arbitrary LTS)

Given an LTS $S = \langle St, L, T, s_0 \rangle$, the following AG G is such that $G = ag(S)$:

$G = \langle Sg, L, Ac, Tg, g_0 \rangle$, where

- (1) $Sg = \{g_i \in P(St) \mid g_i = g_i^\epsilon\}$,
- (2) $g_0 = \{s_i \in St \mid s_0 \xrightarrow{\epsilon} s_i\} (= \{s_i \in St \mid s_0 \xrightarrow{\epsilon} s_i\}^\epsilon) \in Sg$,
- (3) $\forall g_i \in Sg, Ac(g_i) = \{X \mid \exists s_j \in g_i \text{ such that } out(s_j) = X\}$.

- (4) $\forall g_i \in S_g$, we have $g_i \xrightarrow{a} g_j$, iff
 $a \in A$, $A \in Ac(g_i)$ and $g_j = \{s_k \in St \text{ such that } s_j \in g_i \text{ with } s_j \xrightarrow{a} s_k\}^\varepsilon$.

An arbitrary AG G corresponds to a set of equivalent LTSs. However, by Proposition 3.4, for an arbitrary AG G , we define a special LTS S , written $lts(G)$, corresponding to G . For that, each state of G is split into a set of S states as shown in Figure 4. For each non redundant set of interactions A_{ij} , the acceptance set of a state g_i in G corresponds a state $s_{A_{ij}}$ in St . By a non redundant set of interactions, we denote a set of interactions that does not include other sets of interactions in the acceptance set nor it includes a set of interactions that is included in another one. The corresponding S states for a given G state, are defined as follows:

Definition 3.13 (LTS states corresponding to a G state)

Given an AG $G = \langle S_g, L, Ac, \tau, g, g_o \rangle$ and a state g_i in G , the states corresponding to g_i in an LTS corresponding to G are defined as follows:

$$f(g_i) = \{s_{A_{ij}} \mid A_{ij} \in Ac(g_i), \text{ and } A_{ij} \text{ is non redundant, such that } A_{ij} = A_{ik} \text{ or } A_{il} \}$$

Proposition 3.4 (Definition of $lts(G)$ for an arbitrary AG G)

Given an AG $G = \langle S_g, L, Ac, \tau, g, g_o \rangle$,

the following LTS S , written $lts(G)$, is such that $G \equiv ag(S)$:

$S = \langle St, L, T, s_o \rangle$, where

- (1) $St = \bigcup_{g_i \in S_g} (f(g_i))$
- (2) $s_i \xrightarrow{\tau} s_{A_{ij}}$, for each $s_{A_{ij}} \in f(g_i)$, for each s_i in St (see Figure 4),
- (3) For each transition $g_i \xrightarrow{a} g_k$ in G , for each $s_{A_{ij}} \in f(g_i)$, with $a \in A_{ij}$, there is a transition $s_{A_{ij}} \xrightarrow{a} s_k$ in S (see Figure 4).

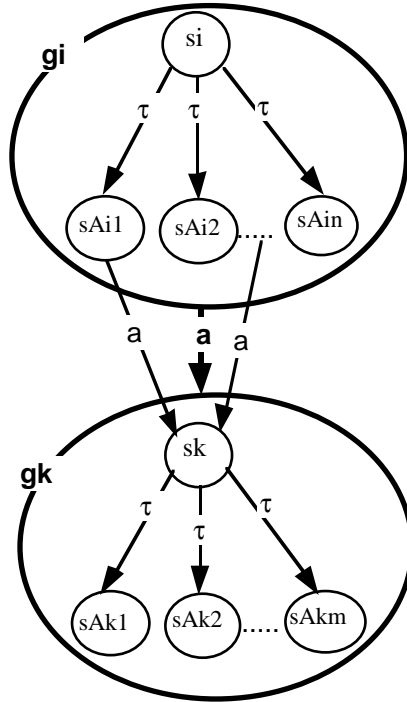


Figure 4. Transformation of the AG G into $lts(G)$.

By definition, for an arbitrary AG G , $lts(G)$ is unique. Due to the special form of LTSs defined by Proposition 3.4, two AGs G_1 and G_2 are (cyclic) bisimulation equivalent, if and only if $lts(G_1)$ and $lts(G_2)$ are (cyclic) strong bisimulation equivalent. Moreover, due to the correspondence between states of an G_1 (resp. G_2) and $lts(G_1)$ (resp. $lts(G_2)$), G_1 and G_2 are isomorphic, if and only if $lts(G_1)$ and $lts(G_2)$ are isomorphic.

Proposition 3.

Given two AGs G_1, G_2 , and two LTSs S_1, S_2 such that $S_1 = lts(G_1)$ and $S_2 = lts(G_2)$, the following holds:

- (1) $S_1 = S_2$ iff $G_1 =_g G_2$,
- (2) $S_1 \sim_c S_2$ iff $G_1 \sim_{c_g} G_2$,
- (3) $lts(G_1) = lts(G_2)$ iff $G_1 =_g G_2$.

For this special form of LTSs, defined in Proposition 3.4, the (cyclic) testing, (cyclic) observation and (cyclic) bisimulation equivalences coincide. Lemma 3.3 follows directly from the facts that $G_1 = ag(lts(G_1))$, $G_2 = ag(lts(G_2))$, Lemma 3.1 and Proposition 3.5.

Lemma 3.3

Given two AGs, G_1 and G_2 ,

(1) the following statements are equivalent:

$$\text{lts}(G1) \text{ te } \text{lts}(G2), \text{ lts}(G1) \approx \text{lts}(G2), \text{ lts}(G1) \text{ lts}(G2), G1 \text{ }_g G2.$$

(2) the following statements are equivalent:

$$\text{lts}(G1) \text{ tec } \text{lts}(G2), \text{ lts}(G1) \approx_c \text{lts}(G2), \text{ lts}(G1) \text{ }_c \text{lts}(G2), G1 \text{ }_{c_g} G2.$$

Note that similar correspondence between LTSs and Tgraphs is used in [Clea 93] in order to verify the testing equivalence relation between LTSs as defined in [Henn 88] by verifying the bisimulation equivalence between the corresponding Tgraphs. Drira has used similar correspondence between LTS and Refusal Graphs for the same purpose as in [Clea 93]. He also defined a special form of LTSs, called normal form, and proved that the testing, observation and bisimulation equivalences coincide for these LTSs, as we have done in the first part of Lemma 3.3. The form of the LTSs defined by Proposition 3.4 is similar to the normal form defined in [Drir 92], except that in our case each state has, in an exclusive manner, transitions labelled by the silent action or transitions labelled by interactions, whereas in [Drir 92] a state may have both kind of transitions.

4 Merging Acceptance Graphs

In this section, we define the merging of AGs. The AGs are more tractable mathematically than the LTSs, because the outgoing transitions, from a given state, have distinct labels. Given two AGs $G1$ and $G2$, we define an operation Merge, such that $\text{Merge}(G1, G2)$ extends $G1$ and $G2$. Moreover, provided that a necessary and sufficient condition holds, $\text{Merge}(G1, G2)$ is the least common cyclic extension of $G1$ and $G2$. The main properties of this Merge operation are described and an algorithm for the construction of $\text{Merge}(G1, G2)$ in the case of FAGs as well as an example of application are given.

4.1 Definition and Properties of the Merge operation

Informally, given two AGs $G1 = \langle Sg1, L1, Ac1, Tg1, g1_o \rangle$ and $G2 = \langle Sg2, L2, Ac2, Tg2, g2_o \rangle$, we define $\text{Merge}(G1, G2)$ to be the reachable part of a graph in which a state g_i is either a pair $\langle g1_i, g2_j \rangle$ consisting of a state $g1_i$ from $Sg1$ and a state $g2_j$ from $Sg2$ (for instance, the initial state $\langle g1_o, g2_o \rangle$), or a simple state $g1_i$ from $Sg1$, or a simple state $g2_j$ from $Sg2$.

The definition of the transitions from a state $\langle g1_i, g2_j \rangle$ in $\text{Merge}(G1, G2)$ depends on the transitions from $g1_i$ in $G1$ and from $g2_j$ in $G2$. For instance, the transition $\langle g1_i, g2_j \rangle \xrightarrow{a} \langle g1_k, g2_m \rangle$ is defined

in $\text{Merge}(G1, G2)$, if and only if there is a transition $g1_i - a \rightarrow g1_k$ in $G1$ and a transition $g2_j - a \rightarrow g2_m$ in $G2$. A transition $\langle g1_i, g2_j \rangle - a \rightarrow g1_k$ is defined in $\text{Merge}(G1, G2)$, if and only if there exist a transition $g1_i - a \rightarrow g1_k$ in $G1$, but there is no transition labeled by a from $g2_j$ in $G2$. The transitions from a simple state in $\text{Merge}(G1, G2)$, such as $g1_k$ for instance, remain the same as defined in $G1$ or $G2$ except for the transitions that reach the initial states of $G1$ or $G2$, which are replaced by corresponding transitions that reach the initial state $\langle g1_o, g2_o \rangle$ of $\text{Merge}(G1, G2)$. A complete definition is as follows:

Definition 4.1 (Merge)

Given two AGs, $G1 = \langle Sg1, L1, Ac1, Tg1, g1_o \rangle$ and $G2 = \langle Sg2, L2, Ac2, Tg2, g2_o \rangle$,

$\text{Merge}(G1, G2) = \text{reachable}(\langle Sg3, L1 \cup L2, Ac3, Tg3, \langle g1_o, g2_o \rangle \rangle)$, where

- (1) $Sg3 = \{ \langle g1_i, g2_k \rangle \mid g1_i \in Sg1 \text{ and } g2_k \in Sg2 \} \cup Sg1 \cup Sg2$
- (2) The mapping $Ac3$ is defined as follows: For each state g_i in $Sg3$,
 - if $g_i = \langle g1_i, g2_j \rangle$, then $Ac3(g_i) = \{ X1 \cup X2 \mid X1 \in Ac1(g1_i) \text{ and } X2 \in Ac2(g2_j) \}$,
 - if $g_i \in Sgx$, then $Ac3(g_i) = Acx(g_i)$, where $x = 1, 2$.
- (3) For each state $\langle g1_j, g2_k \rangle$ in $Sg3$,
 - 3-1. $\langle g1_j, g2_k \rangle - a \rightarrow \langle g1_l, g2_m \rangle \in Tg3$ iff $g1_j - a \rightarrow g1_l \in Tg1$ and $g2_k - a \rightarrow g2_m \in Tg2$.
 - 3-2. $\langle g1_j, g2_k \rangle - a \rightarrow \langle g1_o, g2_o \rangle \in Tg3$ iff $(g1_j - a \rightarrow g1_o \in Tg1 \text{ and } g2_k - /a \rightarrow \text{ in } Tg2)$
or $(g1_j - /a \rightarrow \text{ in } Tg1 \text{ and } g2_k - a \rightarrow g2_o \in Tg2)$.
 - 3-3. $\langle g1_j, g2_k \rangle - a \rightarrow g1_l \in Tg3$ iff $g1_j - a \rightarrow g1_l \in Tg1$, $g1_l \neq g1_o$, and $g2_k - /a \rightarrow \text{ in } Tg2$.
 - 3-4. $\langle g1_j, g2_k \rangle - a \rightarrow g2_m \in Tg3$ iff $g2_k - a \rightarrow g2_m \in Tg2$, $g2_m \neq g2_o$, and $g1_j - /a \rightarrow \text{ in } Tg1$.
- (4) For each state g_{xj} in $Sg3$, where $x = 1, 2$,
 - 4-1. $g_{xj} - a \rightarrow \langle g1_o, g2_o \rangle \in Tg3$ iff $g_{xj} - a \rightarrow g_{xo} \in Tgx$.
 - 4-2. $g_{xj} - a \rightarrow g_{xl} \in Tg3$ iff $g_{xj} - a \rightarrow g_{xl} \in Tgx$, $g_{xl} \neq g_{xo}$.

If we consider, for instance, the AGs $G1$ and $G2$ shown in Figure 5, $\text{Merge}(G1, G2)$ is described by the reachable part (in bold) of G .

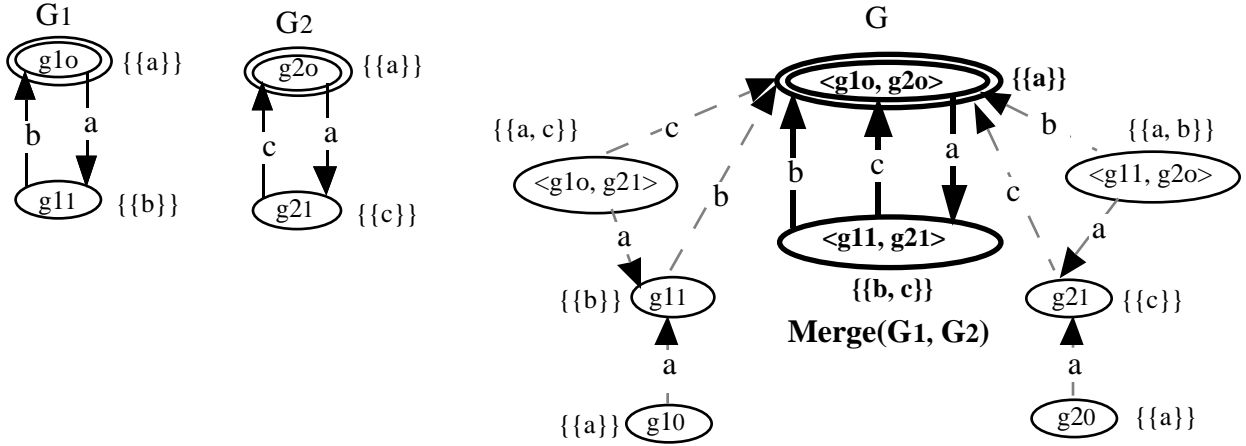


Figure 5. Example of Merge.

Merge(G_1, G_2) defines an AG. The consistency constraints defined in Section 3.1 are satisfied by Merge(G_1, G_2) as stated by Proposition 4.1 below. Stated otherwise, given two AGs G_1 and G_2 , Merge(G_1, G_2), always exists.

Proposition 4.1

Given two AGs, G_1 and G_2 , Merge(G_1, G_2) is an AG.

The operation Merge is commutative and associative. Therefore, AGs may be combined in an incremental way and in any order.

Proposition 4.2

Given three AGs, G_1, G_2 and G_3 , the following holds:

- (a) Merge(G_1, G_2) =_g Merge(G_2, G_1),
- (b) Merge(Merge(G_1, G_2), G_3) =_g Merge(G_1 , Merge(G_2, G_3))

In the remainder of this paper, in order to avoid redundancy whenever G_1 and G_2 play symmetrical roles, we state and prove properties of Merge(G_1, G_1) relatively to G_1 only. Same properties hold with respect to G_2 , since operation Merge is commutative.

Merge(G_1, G_2) always extends G_1 .

Proposition 4.3

Given two AGs, G_1 and G_2 , Merge(G_1, G_2) ext_g G_1 .

In order to be a cyclic extension of $G1$, $\text{Merge}(G1, G2)$ should preserve the cyclic traces of $G1$. $\text{Merge}(G1, G2)$ preserves the cyclic traces of $G1$, if and only if it preserves, at least as cyclic traces, the elementary cyclic traces of $G1$. However, there is some situation where an elementary cyclic trace in $G1$ is a noncyclic trace in $\text{Merge}(G1, G2)$. Indeed, this is the case when a certain elementary cyclic trace σ in $G1$ ($g1_o = \sigma \Rightarrow g1_o$) is a noncyclic trace in $G2$ ($g2_o = \sigma \Rightarrow g2_k$ with $g2_k \neq g2_o$). By definition of Merge, after performing σ , $\text{Merge}(G1, G2)$ reaches a state $\langle g1_o, g2_k \rangle$ different from its initial $\langle g1_o, g2_o \rangle$, since $g2_k \neq g2_o$. Therefore, σ is a noncyclic trace in $\text{Merge}(G1, G2)$. The example in Figure 6 illustrates such situations. For instance, a is an elementary cyclic trace in $G1$ ($g1_o = \sigma \Rightarrow g1_o$), but a is a non cyclic trace in $G2$ ($g2_o = \sigma \Rightarrow g2_1$ with $g2_1 \neq g2_o$). Therefore, a is a non cyclic trace in $\text{Merge}(G1, G2)$ ($\langle g1_o, g2_o \rangle = \sigma \Rightarrow \langle g1_o, g2_1 \rangle$ with $g2_1 \neq g2_o$). In Proposition 4.4, we state a necessary and sufficient condition for an elementary cyclic trace in $G1$ to remain a cyclic trace in $\text{Merge}(G1, G2)$.

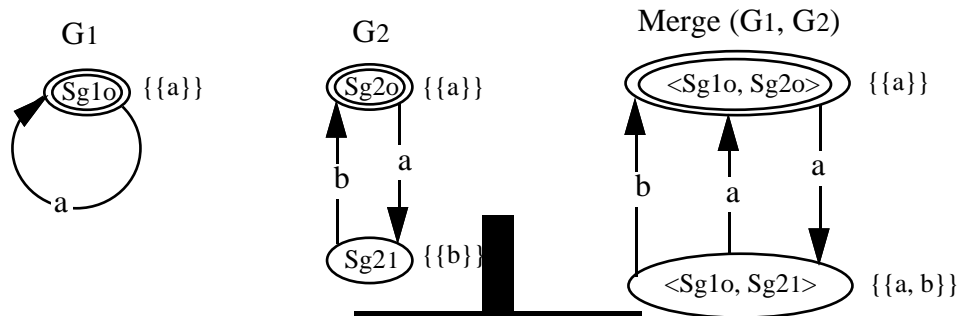


Figure 6. Preservation of cyclic traces by Merge.

Proposition 4.4

Given two AGs, $G1$ and $G2$,
 an elementary cyclic trace σ in $G1$ is a cyclic trace in $\text{Merge}(G1, G2)$
 (σ is a cyclic trace in $G2$ or $\sigma \in \text{Tr}(G2)$).

From Proposition 4.4, it follows that $\text{Merge}(G1, G2)$ preserves the cyclic traces of $G1$ if and only if any elementary cyclic trace σ in $G1$ is a cyclic trace in $G2$ or $\sigma \in \text{Tr}(G2)$, which is equivalent to any elementary cyclic trace σ in $G1$ is a cyclic trace in $G2$ or $\sigma \in \text{Tr}(G2)$ as stated in the following proposition.

Proposition 4.5

Given two AGs, $G1$ and $G2$, the following statements are equivalent:

- (a) $\text{Merge}(G1, G2)$ preserves the cyclic traces of $G1$,
- (b) any elementary cyclic trace σ in $G1$ is a cyclic trace in $G2$ or $\sigma \in \text{Tr}(G2)$.

(c) any cyclic trace σ in G_1 is a cyclic trace in G_2 or $\sigma \in \text{Tr}(G_2)$.

The conditions (b) (and (c)) in Proposition 4.5 can be stated in terms of states as follows: for any state $\langle g_{1i}, g_{2j} \rangle$ in $\text{Merge}(G_1, G_2)$, if $g_{1i} = g_{1o}$ then $g_{2j} = g_{2o}$. This condition is very easy to verify in the case of FAGs.

In Proposition 4.4, we have stated a sufficient and necessary condition for which an elementary cyclic trace σ in G_1 remains a cyclic trace in $\text{Merge}(G_1, G_2)$. Moreover, in this case σ is an elementary cyclic trace in $\text{Merge}(G_1, G_2)$. Indeed, if $\sigma = a_1.a_2\dots a_n$ and $g_{1o} \xrightarrow{a_1} g_{1i}, g_{1i} \xrightarrow{a_2} g_{1i+1} \dots, g_{1i+n-2} \xrightarrow{a_n} g_{1o}$ with $g_{1i+j} = g_{1o}$, for $j = 0, \dots, n-2$, and σ is a cyclic trace in $\text{Merge}(G_1, G_2)$, then by definition of Merge , $\langle g_{1o}, g_{2o} \rangle \xrightarrow{a_1} g_i, g_i \xrightarrow{a_2} g_{i+1} \dots, g_{i+n-2} \xrightarrow{a_n} \langle g_{1o}, g_{2o} \rangle$ with $g_{i+j} = g_{1i+j}$ or $\langle g_{1i+j}, g_{2kj} \rangle$ for some state g_{2kj} in G_2 and $g_{i+j} \in \langle g_{1o}, g_{2o} \rangle$, since $g_{1i+j} = g_{1o}$, for $j = 0, \dots, n-2$. However, an elementary cyclic trace in $\text{Merge}(G_1, G_2)$ is not always an elementary cyclic trace in G_1 or G_2 . As shown by the example in Figure 6, **a.a** is neither an elementary cyclic trace in G_1 nor in G_2 . **a.a** is a cyclic trace in G_1 . As stated by Proposition 4.6, any elementary cyclic trace in $\text{Merge}(G_1, G_2)$ is a cyclic trace in G_1 or G_2 .

Proposition 4.6

Given two AGs, G_1 and G_2 ,
any elementary cyclic trace in $\text{Merge}(G_1, G_2)$ is a cyclic trace in G_1 or G_2 .

Any trace in $\text{Merge}(G_1, G_2)$ results from the recursive concatenation of cyclic traces of G_1 or G_2 , and a certain trace of G_1 or G_2 . In other words, $\text{Merge}(G_1, G_2)$ may only perform what G_1 or G_2 may perform, in a recursive manner.

Proposition 4.7

Given two AGs, G_1 and G_2 ,
any trace σ of $\text{Merge}(G_1, G_2)$ may be written as $\sigma = \sigma_1.\sigma_2\dots\sigma_n.\sigma_{n+1}$, with
 σ_i as a cyclic trace in G_1 or G_2 , for $i = 1, \dots, n$, and ($\sigma_{n+1} \in \text{Tr}(G_1)$ or $\sigma_{n+1} \in \text{Tr}(G_2)$).

In the case where the cyclic traces of G_1 and the cyclic traces of G_2 remain as cyclic traces in $\text{Merge}(G_1, G_2)$, $\text{Merge}(G_1, G_2)$ represents the least common cyclic extension of G_1 and G_2 . The following theorem follows partly from Proposition 4.3 and Proposition 4.5.

Theorem 4.1

Given two AGs, G_1, G_2 ,

Merge(G_1, G_2) is the least common cyclic extension of G_1 and G_2 iff any cyclic trace σ in G_1 is a cyclic trace in G_2 or $\sigma \in \text{Tr}(G_2)$, and reciprocally.

Due to the constraint for the preservation of the cyclic traces of G_1 and G_2 in $\text{Merge}(G_1, G_2)$, bisimulation equivalence is not substitutive under the Merge combinator. In other words, the fact that X is bisimulation equivalent to Y does not ensure that $\text{Merge}(X, Z)$ is bisimulation equivalent to $\text{Merge}(Y, Z)$. The example in Figure 7, for instance, illustrates such situation. We have $G_1 \cong G_3$ but $\text{Merge}(G_1, G_2)$ and $\text{Merge}(G_3, G_2)$ are not bisimulation equivalent. As shown by this example, this is due to the fact that a is a cyclic trace in G_1 but not in G_3 . The cyclic bisimulation equivalence is substitutive under the Merge combinator. As stated by Theorem 4.2, if X is cyclic bisimulation equivalent to Y then $\text{Merge}(X, Z)$ is cyclic bisimulation equivalent to $\text{Merge}(Y, Z)$, for any AG Z . Therefore, $\text{Merge}(X, Z)$ is bisimulation equivalent to $\text{Merge}(Y, Z)$.

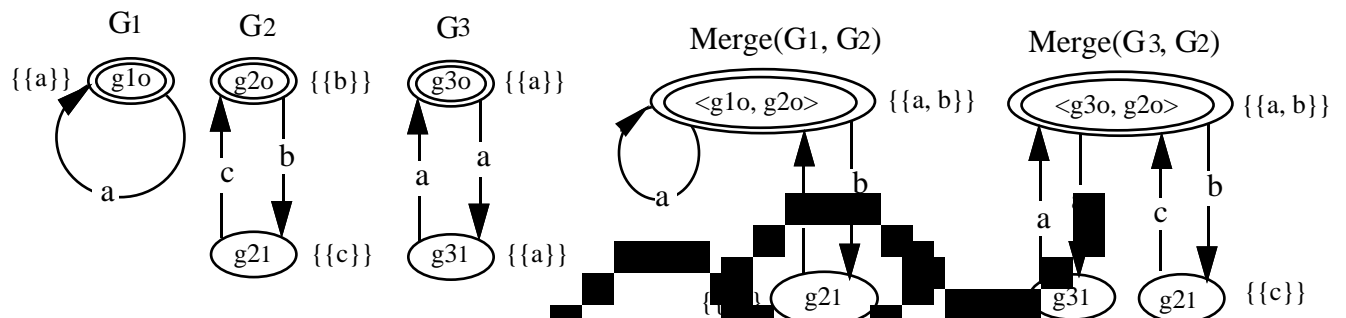


Figure 7. Substitution property of the bisimulation equivalence under Merge.

Theorem 4.2

Given three AGs, G_1 , G_2 , and G_3 , such that $G_1 \cong G_3$, the following holds: $\text{Merge}(G_1, G_2) \cong \text{Merge}(G_3, G_2)$

4.2 Merging FAGs and Application

In the previous section the Merge combinator has been defined for arbitrary AGs. In the following, we describe an algorithm, also called Merge, for the construction of $\text{Merge}(G_1, G_2)$, in the case of FAGs, and we apply it for the combination of two versions of the so-called Daemon Game [ISO 8807]. Notice that, in the case of an FAG G , for any state g_i of G , $\text{Ac}(g_i)$ and any element in $\text{Ac}(g_i)$ are finite, since $\text{Ac}(g_i) \subseteq P(L)$.

Algorithm Merge

Given two AGs, $G1 = \langle Sg1, L1, Ac1, Tg1, g1_0 \rangle$ and $G2 = \langle Sg2, L2, Ac2, Tg2, g2_0 \rangle$,
 $Merge(G1, G2) = \langle Sg3, L1 \cup L2, Ac3, Tg3, g1_0, g2_0 \rangle$, where $Sg3$, $Ac3$ and $Tg3$ are built,
 recursively, as follows:

Initial step:

$Sg3 = \{ \langle g1_0, g2_0 \rangle \}$ and $Ac3(\langle g1_0, g2_0 \rangle) = \{ X1 \cup X2 \mid X1 \in Ac1(g1_0) \text{ and } X2 \in Ac2(g2_0) \}$.

Loop:

For each state g_i entered into $Sg3$ (first for the initial state $\langle g1_0, g2_0 \rangle$) repeat the following:

if $g_i = \langle g1_j, g2_k \rangle$, then for each $A \in Ac3(\langle g1_j, g2_k \rangle)$ and $a \in A$,

if $g1_j - a \rightarrow g1_l \in Tg1$ and $g2_k - a \rightarrow g2_m \in Tg2$, then

$Sg3 = Sg3 \cup \{ \langle g1_l, g2_m \rangle \}$, $Ac3(\langle g1_l, g2_m \rangle) = \{ X1 \cup X2 \mid X1 \in Ac1(g1_l) \text{ and } X2 \in Ac2(g2_m) \}$ and $\langle g1_l, g2_m \rangle - a \rightarrow \langle g1_l, g2_m \rangle \in Tg3$ and

$Ac3(\langle g1_l, g2_m \rangle) = \{ X1 \cup X2 \mid X1 \in Ac1(g1_l) \text{ and } X2 \in Ac2(g2_m) \}$.

if $g1_j - a \rightarrow g1_0 \in Tg1$ and $g2_k - a \rightarrow g2_0 \in Tg2$, then $\langle g1_j, g2_k \rangle - a \rightarrow \langle g1_0, g2_0 \rangle \in Tg3$.

if $g1_j - a \rightarrow g1_0 \in Tg1$ and $g2_k - a \rightarrow g2_m \in Tg2$, then $\langle g1_j, g2_k \rangle - a \rightarrow \langle g1_0, g2_m \rangle \in Tg3$.

if $g1_j - a \rightarrow g1_l \in Tg1$, with $g1_l = g1_0$ and $g2_k - a \rightarrow g2_m \in Tg2$, then

$Sg3 = Sg3 \cup \{ \langle g1_l, g2_m \rangle \}$, $Ac3(\langle g1_l, g2_m \rangle) = \{ X1 \cup X2 \mid X1 \in Ac1(g1_l) \text{ and } X2 \in Ac2(g2_m) \}$ and $\langle g1_l, g2_m \rangle - a \rightarrow g1_l \in Tg3$.

if $g1_j - a \rightarrow g1_0 \in Tg1$ and $g2_k - a \rightarrow g2_m \in Tg2$, with $g2_m = g2_0$, then

$Sg3 = Sg3 \cup \{ \langle g2_m \rangle \}$, $Ac3(\langle g2_m \rangle) = \{ X2 \mid X2 \in Ac2(g2_m) \}$ and $\langle g1_j, g2_k \rangle - a \rightarrow g2_m \in Tg3$.

if $g_i = gx_j$, with $x = 1, 2$, then for each $A \in Ac3(gx_j)$ and $a \in A$,

if $gx_j - a \rightarrow gx_0 \in Tgx$, then $gx_j - a \rightarrow \langle g1_0, g2_0 \rangle \in Tg3$.

if $gx_j - a \rightarrow gx_l \in Tgx$, with $gx_l \neq gx_0$, then

$Sg3 = Sg3 \cup \{ gx_l \}$, $Ac3(gx_l) = \{ X1 \mid X1 \in Ac1(gx_l) \}$ and $gx_j - a \rightarrow gx_l \in Tg3$.

Application

As application, we consider two versions of the Daemon game [ISO 8807]. The first game is called Simple Daemon Game. The player may insert a coin, probe the system, then he randomly loses or wins and collects. The behavior of this game is modeled by the FAG $G1$ in Figure 8 (a). The second game is called Jackpot Daemon Game. The behavior of this second game is as follows: the player has to insert a coin before starting the game. Once the coin has been inserted, the player can probe, then he randomly loses or wins. If he wins, the game continues. He can probe again, then he randomly loses or get the "Jackpot" and collect it. The behavior of Jackpot Daemon Game is modeled by the FAG $G2$ in Figure 8 (b).

Assume that we want to combine these two games, in order to describe a new system, called Combined Game, where the player can, alternatively, play the Simple Daemon Game and the Jackpot Daemon Game, without any interference between these two games. $Merge(G1, G2)$, as shown in Figure 9, defines such a combination of the Simple Daemon Game and the Jackpot

Daemon Game. We have $\text{Merge}(G1, G2)$ extends $G1$ and $G2$. Moreover, any cyclic trace of $G1$ remains as cyclic trace in $\text{Merge}(G1, G2)$, since there is no state $\langle g1_o, g2_j \rangle$ in $\text{Merge}(G1, G2)$ with $g2_j \neq g2_o$. Any cyclic trace of $G2$ remains as cyclic trace in $\text{Merge}(G1, G2)$, since there is no state $\langle g1_i, g2_o \rangle$ in $\text{Merge}(G1, G2)$ with $g1_i \neq g1_o$. $\text{Merge}(G1, G2)$ is the least common cyclic extension of $G1$ and $G2$. $\text{Merge}(G1, G2)$ is able to behave, alternatively, in a recursive manner, as $G1$ and $G2$.

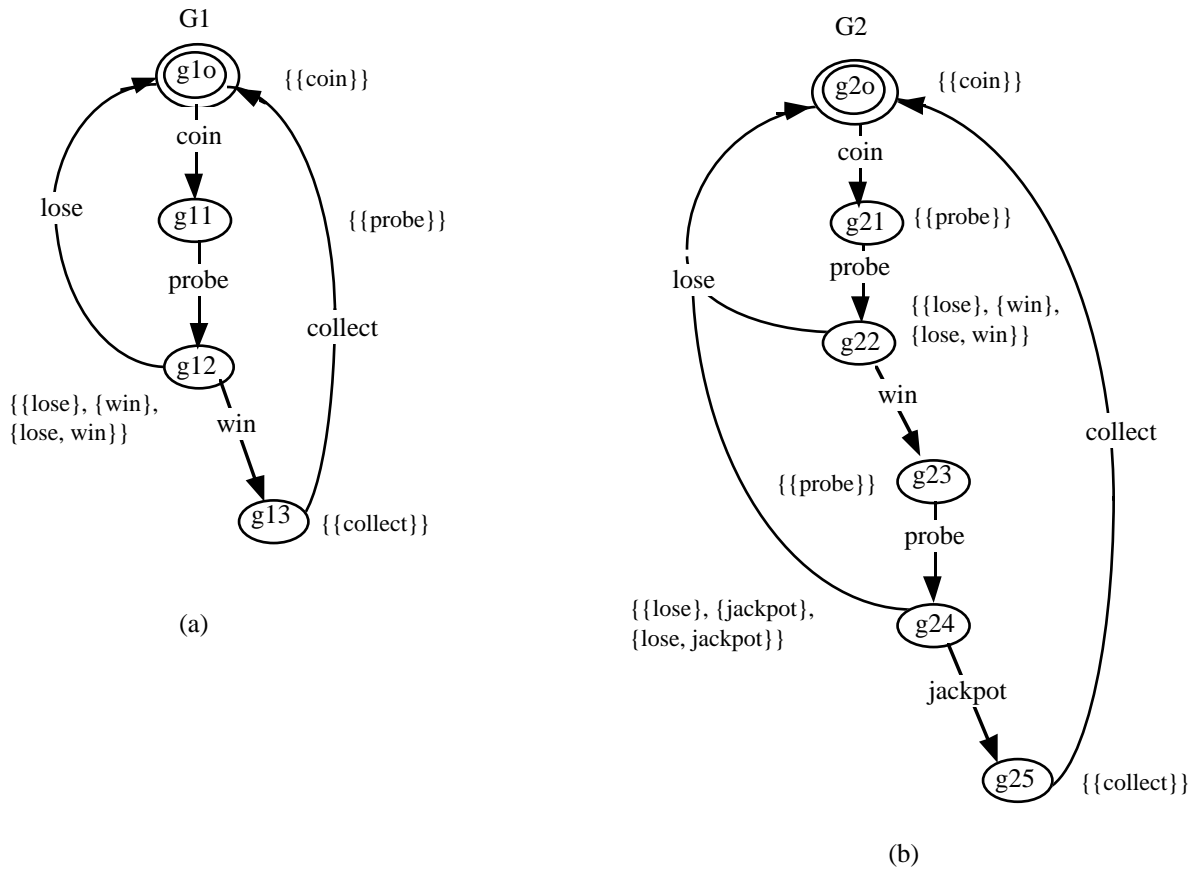


Figure 8. (a) Simple Daemon Game (b) Jackpot Game Descriptions.

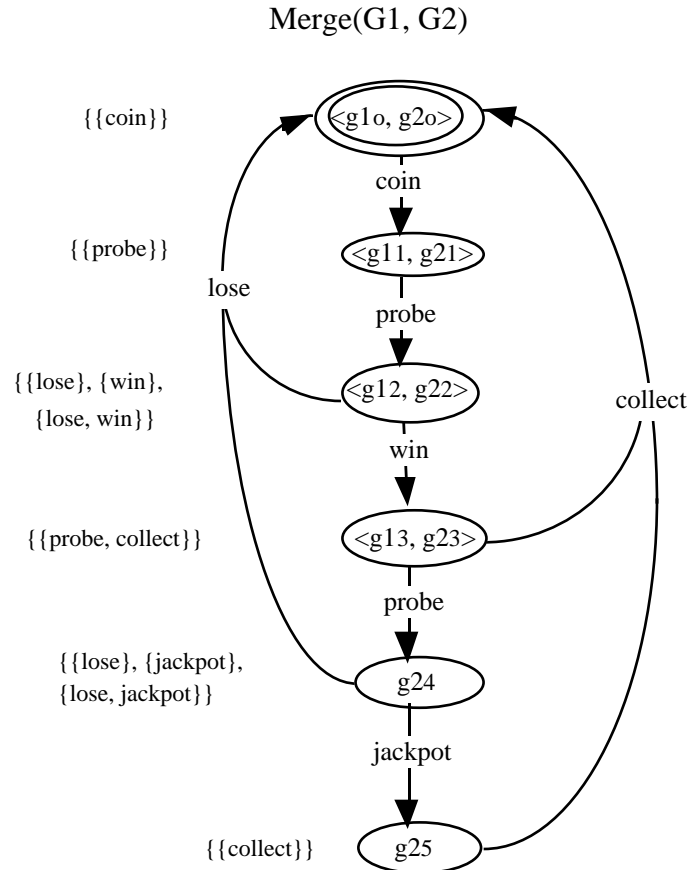


Figure 9. Combined Game Description.

4.3 Discussion

The operation Merge defined in Section 4.1 is such that, for given AGs, G1 and G2, in the case of the cyclic traces of G1 or G2, Merge(G1, G2) may exhibit the behaviors of G1 and the behaviors of G2, in a recursive manner, without any new failure for these behaviors. Consider, for instance, the example in Section 4.2, the Combined Game may exhibit the behaviors of the Simple Daemon Game and the behaviors of the Jackpot Daemon Game, in a recursive manner. Each time the Combined Game exhibits a behavior of the Simple Daemon Game or a behavior of the Jackpot Daemon Game, the Combined Game does not block where the Simple Daemon Game or the Jackpot Daemon Game may not block, respectively.

Merge(G1, G2) always extends G1 and G2. Provided that certain necessary and sufficient condition (Theorem 4.1) is satisfied, Merge(G1, G2) is the least common cyclic extension of G1 and G2. In general, Merge(G1, G2) is not the least common extension of G1 and G2. The least common extension of G1 and G2 is defined by the combinator \oplus , which is very similar to Merge operation, except for the rules defining the transitions, which are replaced by the following rules:

- (3) For each state $\langle g_{1j}, g_{2k} \rangle$ in Sg_3 ,
 - 3-1. $\langle g_{1j}, g_{2k} \rangle \xrightarrow{a} \langle g_{1l}, g_{2m} \rangle \in Tg_3$ iff $g_{1j} \xrightarrow{a} g_{1l} \in Tg_1$ and $g_{2k} \xrightarrow{a} g_{2m} \in Tg_2$.
 - 3-2. $\langle g_{1j}, g_{2k} \rangle \xrightarrow{a} g_{1l} \in Tg_3$ iff $g_{1j} \xrightarrow{a} g_{1l} \in Tg_1$ and $g_{2k} \xrightarrow{a} g_{2m} \in Tg_2$.
 - 3-3. $\langle g_{1j}, g_{2k} \rangle \xrightarrow{a} g_{2m} \in Tg_3$ iff $g_{1j} \xrightarrow{a} g_{1l} \in Tg_1$ and $g_{2k} \xrightarrow{a} g_{2m} \in Tg_2$.
- (4) For each state g_{xj} in Sg_3 where $x = 1, 2$,
 - 4-1. $g_{xj} \xrightarrow{a} g_{xi} \in Tg_3$ iff $g_{xj} \xrightarrow{a} g_{xi} \in Tg_x$.

Contrarily to Merge(G_1, G_2), in $G_1 \otimes G_2$, the initial state g_{10} of G_1 (respectively g_{20} of G_2), that reaches the initial state g_{10} (respectively g_{20}) of G_1 (respectively G_2) is preserved without change. For a state $\langle g_{1j}, g_{2k} \rangle$ in Sg_3 , if $g_{1j} \xrightarrow{a} g_{1l}$ in G_1 (instead of $\langle g_{1j}, g_{2k} \rangle \xrightarrow{a} \langle g_{1l}, g_{2k} \rangle$ in $G_1 \otimes G_2$), then the transition $g_{1j} \xrightarrow{a} g_{1l}$ is preserved in $G_1 \otimes G_2$. Dually, if the initial state g_{20} of G_2 (respectively g_{20} of G_2) is reached, then G_2 behaves as if it were alone.

The combinator \otimes defines an AG, is commutative and associative. $G_1 \otimes G_2$ always extends G_1 and G_2 . In fact, $G_1 \otimes G_2$ is the least common extension of G_1 and G_2 . By definition, $G_1 \otimes G_2$ does not preserve the cyclic traces of G_1 (respectively G_2), except the cyclic traces common to G_1 and G_2 . We have $Tr(G_1 \otimes G_2) = Tr(G_1) \cap Tr(G_2)$. $G_1 \otimes G_2$ describes the behaviors of G_1 and G_2 in parallel. The example in figure 10, which describes the least common extension of the Simple Daemon Game and the Jackpot Daemon Game, illustrates this property. G_1 describes a system which may be only as the Simple Daemon Game and the Jackpot Daemon Game.

The operation \otimes preserves the bisimulation equivalence. In other words, if $G_1 \sim G_2$ then $G_1 \otimes G_3 \sim G_2 \otimes G_3$. Moreover, \otimes preserves the cyclic bisimulation equivalence, since only the common cyclic traces of G_1 (respectively G_3) and G_2 are preserved in $G_1 \otimes G_2$ (respectively $G_3 \otimes G_2$).

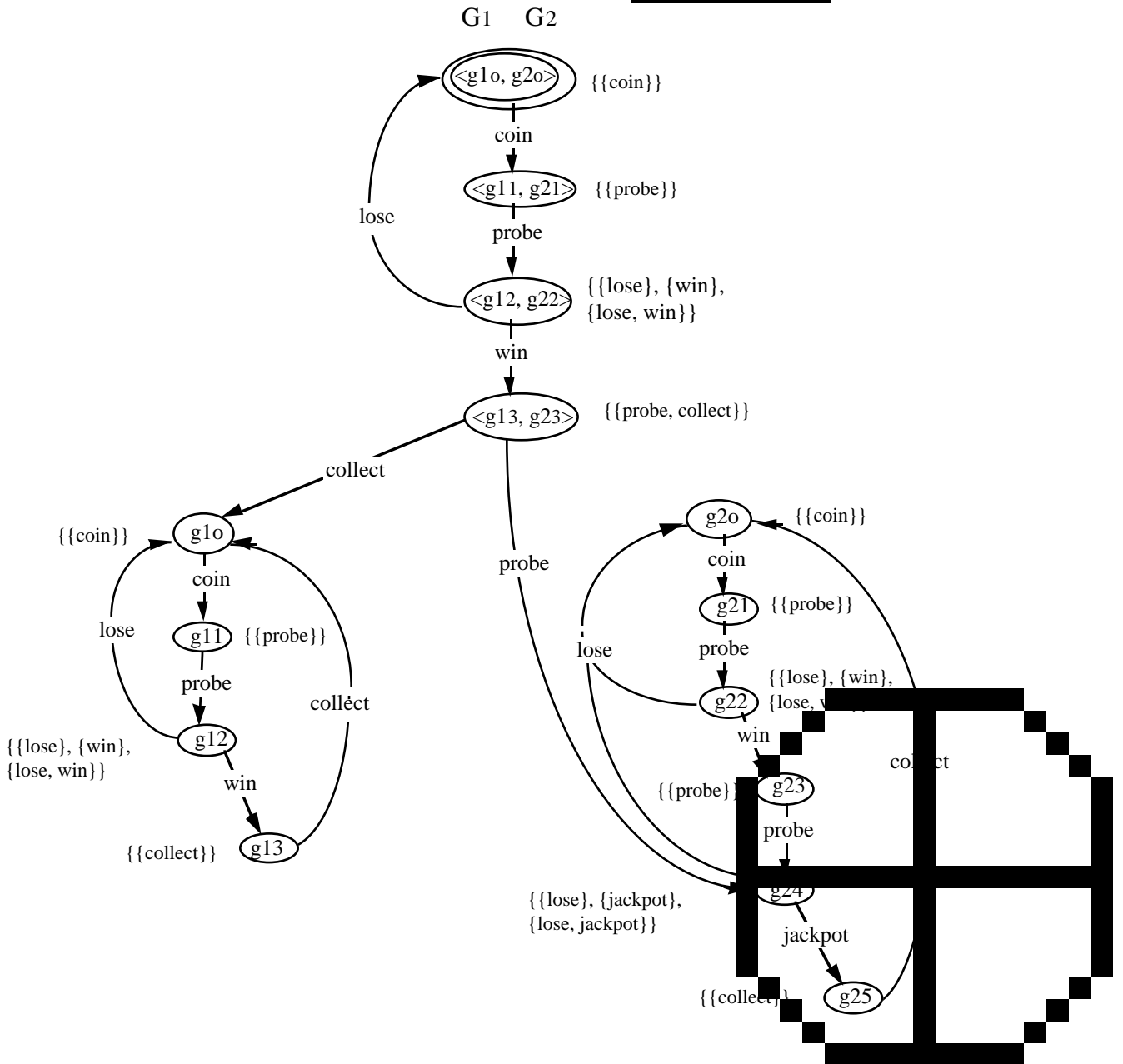


Figure 10. Application of the operation .

5 Merging Labelled Transition Systems

The definition of Merge for LTSs is based on the definition of Merge for AGs and the correspondence between LTSs and AGs.

5.1 Definition and Properties of Merge

Definition 5.1 (Merge for LTSs)

Given two LTSs $S1$ and $S2$, $Merge(S1, S2) = lts(Merge(ag(S1), ag(S2)))$.

Since for any LTS S , there is one and only one AG G such that $G = ag(S)$, for any AG G there is one and only one LTS such $S = lts(G)$, and for given AGs $G1$ and $G2$, $Merge(G1, G2)$ always exists and uniquely defined, then for given LTSs $S1$ and $S2$, $Merge(S1, S2)$, always, exists and is uniquely defined.

All the propositions, lemmas and Theorem 4.1 stated for Merge in the case of AGs holds for Merge in the case of LTSs. For instance, $Merge(S1, S2)$ always extends $S1$ and $S2$. $Merge(S1, S2)$ is commutative and associative. $Merge(S1, S2)$ is the least common cyclic extension of $S1$ and $S2$, if and only if any cyclic trace σ in $S1$ is a cyclic trace in $S2$ or $\sigma \in Tr(S2)$ and reciprocally.

By correspondence to the AGs and Theorem 4.2, the testing, observation, strong bisimulation equivalences are not substitutive under the LTSs Merge combinator. However, the cyclic (testing, observation, strong bisimulation) equivalences are substitutive under the LTSs Merge combinator. The fact that X and Y are, at least, cyclic testing equivalent ensures that $Merge(X, Z)$ is cyclic bisimulation equivalent to $Merge(Y, Z)$. Indeed, if X and Y are, at least, cyclic testing equivalent, their corresponding AGs $ag(X)$ and $ag(Y)$ are bisimulation equivalent (Lemma 3.1 in Section 3), $Merge(ag(X), ag(Z))$ is cyclic bisimulation equivalent to $Merge(ag(Y), ag(Z))$ (Section 4), and $lts(Merge(ag(X), ag(Z)))$ and $lts(Merge(ag(Y), ag(Z)))$ are cyclic bisimulation equivalent (Proposition 3.5 in Section 3).

Similarly to Merge, $S1 \sqcup S2 = lts(ag(S1) \sqcup ag(S2))$. By correspondence to the AGs, $S1 \sqcup S2$ is the least common extension of $S1$ and $S2$ and the properties of \sqcup in the case of AGs hold for \sqcup in the case of LTSs.

5.2 Merging FLTSs and Application

In the previous section, we defined $Merge(S1, S2)$ for arbitrary LTSs. In this section, we describe an algorithm for the construction of $Merge(S1, S2)$, for the case where $S1$ and $S2$ are FLTSs. This algorithm consists of three steps. In the first step, $S1$ and $S2$ are transformed into FAGs $G1$ and $G2$, such that $G1 = ag(S1)$ and $G2 = ag(S2)$. In the second step, $Merge(G1, G2)$ is constructed

following algorithm Merge described in Section 4.2. In the last step, Merge(G1, G2) is translated into lts(Merge(G1, G2)).

5.2.1 From an FLTS to an FAG

Given an FLTS $S = \langle St, L, T, s_0 \rangle$, the following algorithm derives the corresponding FAG $G = \langle Sg, L, Ac, Tg, g_0 \rangle$. It is based on the "subset construction" algorithm defined in [Hopc 79].

Step 1: Apply the "subset construction" algorithm [Hopc 79], which transforms a nondeterministic finite state automata to a deterministic one (in our case G). To each state in G corresponds a set of states in S . To the state g_0 , for instance, corresponds the set of states $\{s_i \in St \mid s_0 \xrightarrow{\epsilon} s_i\}$.

Step 2: For each state g_i in G , $Ac(g_i) = \{X \mid \text{out}(s_j) = X \text{ for some } s_j \in \{s_1, s_2, \dots, s_m\} \text{ corresponds to } g_i\}$.

5.2.2 From an FAG to an FLTS

Given an FAG $G = \langle Sg, L, Ac, Tg, g_0 \rangle$, the following algorithm allows to derive the FLTS $S = \langle St, L, T, s_0 \rangle = \text{lts}(G)$.

Step 1: (Reduction of the acceptance sets):
 $\forall g_i \in Sg, Ac'(g_i) = \{X \mid X \in Ac(g_i), \text{ such that } Y \text{ and } X = Y \text{ or } Y$

Step 2: Each state g_i is decomposed into $k+1$ LTS states $s_i, s_{i1}, s_{i2}, \dots, s_{ik}$, where $k = \text{cardinal}(Ac'(g_i))$. s_0 represents the initial state of S . Each state s_{ij} corresponds to an element A_{ij} of $Ac'(g_i)$.
 The transitions $s_i \xrightarrow{\tau} s_{ij}$ are defined in S , for $j=1, \dots, k$, for each state s_i in S .

Step 3: For each state s_{ij} in St , for each $a \in A_{ij}$, if $g_i \xrightarrow{a} g_m \in Tg$, then $s_{ij} \xrightarrow{a} s_m \in T$.

5.2.3 Application

We consider the same example as in Section 4. The behaviors of the "Simple Daemon Game" and the "Jackpot Daemon Game" are modeled by FLTSs S1 and S2 in Figure 11, respectively. Merging S1

and S2 yields the FLTS S3 shown in Figure 11. S3 extends S1 and S2. Moreover, any cyclic trace of S1 or S2 remains a cyclic trace in S3. S3 is the least common cyclic extension of S1 and S2. S3 may behave, alternatively, in a recursive manner, as S1 and S2. Note that S3 may be reduced with respect to the (cyclic) observation equivalence by removing some internal transitions τ .

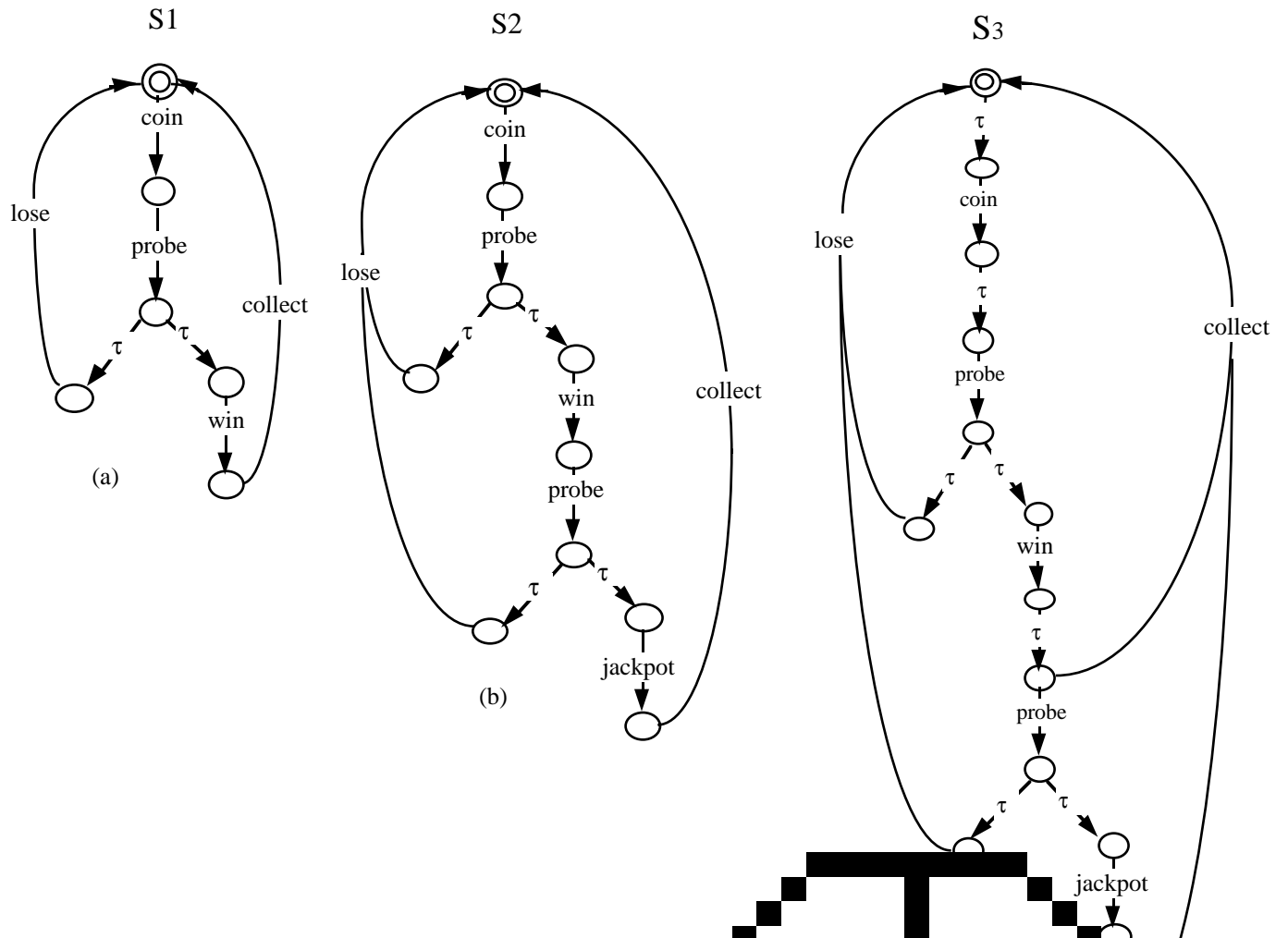


Figure 11. (a) Simple Daemon Game (b) Jackpot Game

6 Related work

In [Ichi 90], the problem of incremental specification in the LOTOS specification language is approached. They introduced a new LOTOS operator and defined the corresponding inference rules, called specification merging operator. This approach is restricted to behavior specifications without the internal action τ . B1 B2 defines a behavior, which is supposed to be an extension of

B1 and B2. Unfortunately, it is not always the case as shown by the counter-example of Figure 12. For instance, B1 never refuses interaction c after trace a.b, whereas B1 ⊕ B2 may refuse interaction c after trace a.b. Moreover, B1 ⊕ B2 is not able to behave, alternatively, as B1 and B2. B1 ⊕ B2 may behave only as B1 or only as B2, once the environment has chosen B1 or B2, respectively. In the case of deterministic LTSs, this combinator leads the same LTS as the combinator (merging without taking into account the preservation of cyclic traces) introduced in this paper.

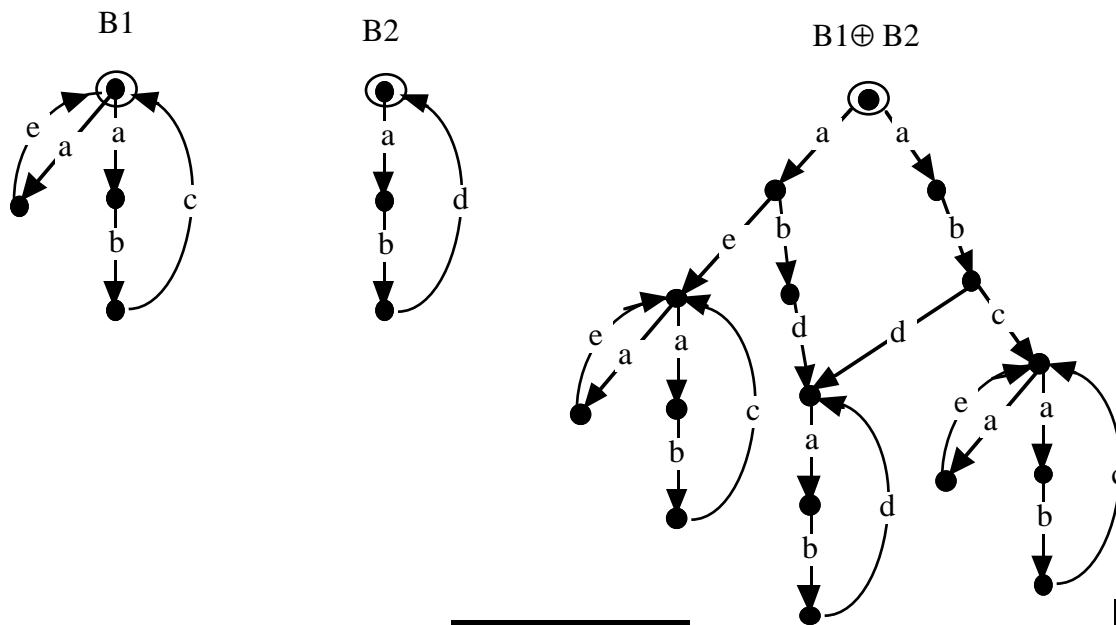


Figure 12. Counter-example for the merging operator.

Mayr has considered the choice operator of the LOTOS language for the extension of behavior specifications [Mayr 88]. The extension of a behavior t by a behavior m is denoted by $t \oplus m$. However, strong restrictions are imposed in order to ensure that s extends t. For instance, the initial interactions of m should be distinct from initial interactions of t.

In [Rudk 91] the notion of inheritance is defined for LOTOS. It is seen as an incremental modification technique. A corresponding operator is introduced and denoted by \oplus . This operator is defined such that if $s = t \oplus m$, then s extends t and any recursive call in t or m is redirected to s. However, strong restrictions are imposed on t and m, such that m should be stable (no internal transition as first event), the initial events of m should be unique and distinct from initial events of t, and so on. The specifications B1 and B2 in Figure 14, for instance, do not satisfy such requirements. In order to define a recursive choice between t and m, Rudkin extended the LOTOS language by a new primitive process "self". There is no requirement such that s should also extend

m, and no considerations to the structure of t or how this modification m is propagated to the processes in t.

Lin has developed an approach for merging alternative protocol functions [Lin 91]. The approach is based on the model of communicating finite state machines. It consists of designing a component protocol for each individual function and then combine them into a single alternating-function protocol. The combination algorithm resolves problems of competition and synchronization between the component protocols, in order to preserve the safety properties (absence deadlock and unspecified receptions) of the component protocols. However, this approach does not take into account the service realized by each protocol component and how this service is preserved in the alternating-function protocol.

7 Conclusion

In this paper, we described an approach for merging behavior specifications. These behaviors are modeled by acceptance graphs or labelled transition systems. Given two behavior specifications B_1 and B_2 , we defined the merging of B_1 and B_2 , written $\text{Merge}(B_1, B_2)$. We proved certain properties of Merge ; for instance, $\text{Merge}(B_1, B_2)$ extends B_1 and B_2 . Provided that a necessary and sufficient condition holds, the cyclic traces in B_1 (respectively B_2) remain cyclic traces in $\text{Merge}(B_1, B_2)$. Therefore, $\text{Merge}(B_1, B_2)$ is a cyclic extension of B_1 and B_2 . Moreover, in this case, $\text{Merge}(B_1, B_2)$ is the least common cyclic extension of B_1 and B_2 . We defined a second combinator, Join , which is very similar to Merge , but differs on the treatment of the cyclic traces of B_1 and B_2 . The operation Join always leads the least common extension of B_1 and B_2 .

The proposed approach for merging behavior specifications is useful for the construction of multiple-function specifications. Instead of handling all the functions simultaneously, the designer may design and verify one function at a time. The merging approach will then derive the required combined specification. From another point of view, it allows the designer to enrich existing specifications with new behaviors required by the user and to integrate existing system specifications.

The approach introduced in this paper has been extended to structured specifications, i.e. specifications which are modeled as parallel composition of subsystem specifications [Khen 93]. As future development, the application of the extended approach to real case system specifications, such as the telephone system specification, is expected.

The labelled transition systems model used in this paper is the underlying semantic model for many specification languages, such as LOTOS [ISO 8807] and CCS [Milne 91]. The formal examination of the algebraic properties of the merging operators Merge and \parallel as well as the congruence property of the newly introduced (cyclic) equivalences in the context of these languages is left for future development.

References

- [Brin 86] E. Brinksma, G. Scollo and S. Steenbergen, LOTOS specifications, their implementations and their tests, Protocol Specification, testing, and verification, Montréal, Canada, June 1986, Sarikaya and Bochmann (eds.).
- [Broo 85] S. D. Brookes et A. W. Roscoe, An Improved Failure Model for Communicating Sequential Processes, Proceedings of the NSF-SERC Seminar on Concurrency, Springer-Verlag LNCS 197, 1985.
- [DeNi 84] R. De Nicola et M. Hennessy, Testing equivalences for processes, Theo. Comp. Sci. 34, 1984, pp. 83-133.
- [Deni 87] R. De Nicola, Extensional Equivalences for Transition Systems, Acta Informatica, 24, 1987, pp. 211-237.
- [Clea 93] R. Cleaveland and M. Hennessy, Testing Equivalence as Bisimulation Equivalence, Formal Aspects of Computing, 5, pp. 1-20, 1993.
- [Drira 92] K. Drira, Transformation et composition de graphes de refus: analyse de la testabilité, Doctorat Thesis, Université de Toulouse, 1992.
- [Henn 85] M. Hennessy, Acceptances Trees, J. of ACM, Vol.32, No. 4, Oct. 85, pp. 896 - 928.
- [Henn 88] M. Hennessy, Algebraic Theory for Processes, MIT Press, Cambridge, 1988.
- [Hopc 79] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979, 418p.

- [Ichi 90] H. Ichikawa, K. Yamanaka and J. Kato, Incremental Specification in LOTOS, Symposium on Protocol Specification, Testing and Verification X (1990), Ottawa, Canada, Logrippo, Probert and Ural (eds.).
- [ISO 8807] ISO -Information Processing Systems - Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, DIS 8807, 1987.
- [Khen 93] F. Khendek and G.v. Bochmann, Incremental Construction Approach for Distributed System Specifications, Proceedings of the Int. Symp. on Formal Description Techniques, Boston, Mas., 26-29 Oct., 1993.
- [Ledu 90] G. Leduc, On the role of Implementation Relations in the Design of Distributed systems using LOTOS, Doctoral Dissertation, Liège, Belgium.
- [Kell 76] R. Keller, Formal verification of parallel programs, Comm. of the ACM 19, July 1976, pp. 371-384.
- [Lin 91] H. A. Lin, Constructing Protocols with Alternative Functions, IEEE Transactions on Computer, Vol. 40, No. 4, April 1991.
- [Mayr 88] T. Mayr, Specification of object-oriented systems in LOTOS, FORTE, Stirling, 1988.
- [Miln 89] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
- [Park 81] D. Park, Concurrency and Automata in Infinite Strings, Lecture Notes in Computer Science 104, Springer-Verlag, Berlin, 1981, pp. 167-183.
- [Rudk 91] S. Rudkin, Inheritance in LOTOS, Formal description technique - FORTE, Sydney, Australia, 1991, pp. 415 - 430.

Appendix

Proposition 3.1

Consider two AGs, $G_1 = \langle Sg_1, L_1, Ac_1, Tg_1, g_{1_0} \rangle$ and $G_2 = \langle Sg_2, L_2, Ac_2, Tg_2, g_{2_0} \rangle$.

1 - Assume that $\text{Tr}(G1) = \text{Tr}(G2)$ and $(\forall \sigma \in \text{Tr}(G1), \text{Ac1}(g1_0 \text{ after } \sigma) = \text{Ac2}(g2_0 \text{ after } \sigma))$.

To prove that $G1 \sim_g G2$, we have to prove that the relation $\{(g1_0 \text{ after } \sigma), (g2_0 \text{ after } \sigma) : \sigma \in \text{Tr}(G1)\}$ is a bisimulation. By hypothesis, $\text{Ac1}(g1_0 \text{ after } \sigma) = \text{Ac2}(g2_0 \text{ after } \sigma), \forall \sigma \in \text{Tr}(G1)$.

Consider $(g1_0 \text{ after } \sigma) \xrightarrow{a} g1_i$, for some $\sigma \in \text{Tr}(G1)$, $(g1_0 \text{ after } \sigma) \xrightarrow{a} g1_i$, if and only if $(g2_0 \text{ after } \sigma) \xrightarrow{a} g2_j$, since $\text{Tr}(G1) = \text{Tr}(G2)$. We have $g1_i = g1_0 \text{ after } \sigma.a$ and $g2_j = g2_0 \text{ after } \sigma.a$, since the transition relation is a function in the case of AGs. Therefore, $(g1_i, g2_j) \in \{(g1_0 \text{ after } \sigma, g2_0 \text{ after } \sigma) : \sigma \in \text{Tr}(G1)\}$, and the relation $\{(g1_0 \text{ after } \sigma), (g2_0 \text{ after } \sigma) : \sigma \in \text{Tr}(G1)\}$ is a bisimulation.

2 - $G1 \sim_g G2$, there is a bisimulation R such that $(g1_0, g2_0) \in R$, and $\forall (g1_i, g2_j) \in R, \text{Ac1}(g1_i) = \text{Ac2}(g2_j)$.

Consider σ , an arbitrary sequence of actions. First case $\sigma = \epsilon$, it is obvious that $\epsilon \in \text{Tr}(G1)$ and $\epsilon \in \text{Tr}(G2)$. By definition of AGs, $g1_0 \text{ after } \epsilon = g1_0$ and $g2_0 \text{ after } \epsilon = g2_0$. By hypothesis, $(g1_0, g2_0) \in R$ and $\text{Ac1}(g1_0 \text{ after } \epsilon) = \text{Ac2}(g2_0 \text{ after } \epsilon)$. Second case $\sigma = a1.a2\dots an, \sigma \in \text{Tr}(G1)$ if and only if $g1_0 \xrightarrow{a1} g1_{i1} \xrightarrow{a2} g1_{i2} \dots g1_{in-1} \xrightarrow{an} g1_{in}$. The transition relations $Tg1$ and $Tg2$ are functions and $(g1_0, g2_0) \in R$. It follows that $g1_0 \xrightarrow{a1} g1_{i1} \xrightarrow{a2} g1_{i2} \dots g1_{in-1} \xrightarrow{an} g1_{in}$ if and only if $g2_0 \xrightarrow{a1} g2_{j1} \xrightarrow{a2} g2_{j2} \dots g2_{jn-1} \xrightarrow{an} g2_{jn}$ with $(g1_{i1}, g2_{j1}) \in R, (g1_{i2}, g2_{j2}) \in R, \dots$ and $(g1_{in-1}, g2_{jn-1}) \in R$. Consequently, $\sigma \in \text{Tr}(G1)$ if and only if $\sigma \in \text{Tr}(G2)$ ($\text{Tr}(G1) = \text{Tr}(G2)$) and $\text{Ac1}(g1_0 \text{ after } \sigma) = \text{Ac2}(g2_0 \text{ after } \sigma)$.

Proposition 3.2

Consider the AGs, $G1, G2$ and the LTSs $S1, S2$ with $g1_0, g2_0, s1_0, s2_0$, as in figure 3.1.

1- First, we have to prove that $S2 \text{ ext } S1 \iff G2 \text{ ext}_g G1$.

1 - 1 - Prove that $S2 \text{ ext } S1 \Rightarrow G2 \text{ ext}_g G1$:

1 - 1 - a - Prove that $\text{Tr}(G1) = \text{Tr}(G2); G1 = \text{ag}(S1)$ implies that $\text{Tr}(S1) = \text{Tr}(G1)$. $G2 = \text{ag}(S2)$ implies that $\text{Tr}(S2) = \text{Tr}(G2)$. $S2 \text{ ext } S1$ implies that $\text{Tr}(S1) = \text{Tr}(S2)$.

1 - 1 - b - $\forall \sigma \in \text{Tr}(G1), \text{Ac2}(g2_0 \text{ after } \sigma) = \text{Ac1}(g1_0 \text{ after } \sigma)$: $G1 = \text{ag}(S1)$ implies that $\text{Ac1}(g1_0 \text{ after } \sigma) = \text{Acc}(s1_0, \sigma)$. $G2 = \text{ag}(S2)$ implies that $\text{Ac2}(g2_0 \text{ after } \sigma) = \text{Acc}(s2_0, \sigma)$. $\forall \sigma \in \text{Tr}(S1)$, $\text{Acc}(s1_0, \sigma) = \text{Acc}(s2_0, \sigma)$, because $S2 \text{ ext } S1$. It follows that, $\forall \sigma \in \text{Tr}(G1), \text{Ac2}(g2_0 \text{ after } \sigma) = \text{Ac1}(g1_0 \text{ after } \sigma)$. Consequently, $S2 \text{ ext } S1 \Rightarrow G2 \text{ ext}_g G1$.

1 - 2 - The proof for $G2 \text{ ext}_g G1 \Rightarrow S2 \text{ ext } S1$ is very similar.

2 - Any cyclic trace in $S1$ is a cyclic trace in $S2$, iff any cyclic trace in $G1$ is a cyclic trace in $G2$:

2 - 1 - Any cyclic trace in $S1$ is a cyclic trace in $S2 \Rightarrow$ any cyclic trace in $G1$ is a cyclic trace in $G2$:

$G1 = \text{ag}(S1)$, it follows that any cyclic trace in $S1$ is a cyclic trace in $G1$, and reciprocally.

$G2 = \text{ag}(S2)$, it follows that any cyclic trace in $S2$ is a cyclic trace in $G2$, and reciprocally.

Now, assume that any cyclic trace in $S1$ is a cyclic trace in $S2$. It follows that any cyclic trace in $G1$ is a cyclic trace in $S2$. We deduce that any cyclic trace in $G1$ is a cyclic trace in $G2$, which concludes the

first part of the proof. The proof for any cyclic trace in G_1 is a cyclic trace in $G_2 \Rightarrow$ any cyclic trace in S_1 is a cyclic trace in S_2 is similar.

Proposition 3.3

Consider an LTS $S = \langle St, L, T, s_0 \rangle$ and the graph $G = \langle Sg, L, Ac, Tg, g_0 \rangle$ defined by Proposition 3.3. We first have to prove that G is an AG. The constraints C_0, C_3, C_4 are satisfied by definition of $Ac(g_i)$, for each state g_i in Sg . Constraint C_2 is satisfied by definition of the transitions in G . We have to prove that G satisfies constraint C_1 : Given a state g_i , we have to prove that $\forall a \in A, A \in Ac(g_i)$, there is one and only one g_j such that $g_i \xrightarrow{a} g_j$: by definition of $G, \forall a \in A, \text{ and } A \in Ac(g_i), g_i \xrightarrow{a} g_j$ iff $g_j = \{s_j \in St \mid \exists s_m \in g_i \text{ such that } s_m \xrightarrow{a} s_j\}^\varepsilon$. $\forall a \in A, \text{ and } A \in Ac(g_i), g_j$ always exists, since $\forall a \in L, a \in A, \text{ and } A \in Ac(g_i)$, if and only if there exists at least one state s_k in g_i such that $s_k \xrightarrow{a}$ (or a state s_m such that $s_m \xrightarrow{a}$). $g_j = \{s_j \in St \mid \exists s_m \in g_i \text{ such that } s_m \xrightarrow{a} s_j\}^\varepsilon$ is unique, because the set $\{s_j \in St \mid \exists s_m \in g_i \text{ such that } s_m \xrightarrow{a} s_j\}$ is unique.

The proof of $G = ag(S)$ follows directly from the definition of G , it is clear that $g_0 = \sigma \Rightarrow g_i$, iff $g_i = (s_0 \text{ after } \sigma)$. It follows that $Tr(g_0) = Tr(s_0)$ and from the definition of Ac for each state in $Sg, \forall \sigma \in Tr(g_0)$, with $g_0 = \sigma \Rightarrow g_i, Ac(g_i) = Acc(s_0, \sigma)$. For the cyclic traces, from the definition of G we have, $\forall \sigma \in Tr(g_0), g_0 = \sigma \Rightarrow g_0$ iff $(s_0 \text{ after } \sigma) = g_0 = \{s_i \in St \text{ such that } s_0 \xrightarrow{\varepsilon} s_i\}$, it follows that a trace σ is a cyclic trace in G , iff σ is a cyclic trace in S .

Proposition 3.4

Consider an AG $G = \langle Sg, L, Ac, Tg, g_0 \rangle$ and the LTS $S = \langle St, L, T, s_0 \rangle = lts(G)$ as defined by Prop. 3.4. A trace $\sigma \in Tr(s_0)$ iff there is a state s_i such that $s_0 = \sigma \Rightarrow s_i$. From the definition of S , the state s_i exists iff there is a state g_i in G such that $g_0 = \sigma \Rightarrow g_i$. It follows that $Tr(G) = Tr(S)$.

By definition of $S, (s_0 \text{ after } \sigma) = \{s_i \mid s_0 \xrightarrow{\varepsilon} s_i\}$ iff $g_0 = \sigma \Rightarrow g_i$, iff $g_i \in Ac(g_0)$. From the definition of the transitions in $S, s_{Ak} \xrightarrow{a} s_0$ iff $g_k \xrightarrow{a} g_0$. Moreover, in this case, there is no transition $s_{Ak} \xrightarrow{a} s_0$ in S . It follows that $(s_0 \text{ after } \sigma) = \{s_i \mid s_0 \xrightarrow{\varepsilon} s_i\}$ iff $g_0 = \sigma \Rightarrow g_i$.

Proposition 3.5

Consider the AGs $G_1 = \langle Sg_1, L_1, Ac_1, Tg_1, g_{1_0} \rangle, G_2 = \langle Sg_2, L_2, Ac_2, Tg_2, g_{2_0} \rangle$, and the LTSs $S_1 = \langle S_1, L_1, T_1, s_{1_0} \rangle, S_2 = \langle S_2, L_2, T_2, s_{2_0} \rangle$, such that $S_1 = lts(G_1)$ and $S_2 = lts(G_2)$. $S_1 \sim S_2$ implies that $S_1 \text{ te } S_2$. By Lemma 3.1 it follows that $G_1 \sim G_2$, since $G_1 = ag(S_1)$ and $G_2 = ag(S_2)$.

1 - b - $G1 \cong G2$: by definition, we have $G_i = \text{ag}(\text{Lts}(S_i))$, $i = 1, 2$. It follows that $\text{Tr}(S_i) = \text{Tr}(G_i)$, $i = 1, 2$.
 By hypothesis, $G1 \cong G2$, therefore $\text{Tr}(S1) = \text{Tr}(S2) = \text{Tr}(G1) = \text{Tr}(G2)$. We have to prove that the following relation $R = \{(s1_i, s2_j): s1_o = \sigma \Rightarrow s1_i \xrightarrow{\tau}, s2_o = \sigma \Rightarrow s2_j \xrightarrow{\tau}, \sigma \in \text{Tr}(S1)\} (= R1) \cup \{(s1_{Aik}, s2_{Ajl}): s1_{Aik} \in f(g1_o \text{ after } \sigma), s2_{Ajl} \in f(g2_o \text{ after } \sigma), Aik = Ajl, \text{ and } \sigma \in \text{Tr}(S1)\} (= R2)$ is a strong bisimulation. Note that $(s1_o, s2_o) \in R1$.

- Consider an element $(s1_i, s2_j) \in R1$. By definition of $R1$, for some $\sigma \in \text{Tr}(S1)$, $s1_o = \sigma \Rightarrow s1_i \xrightarrow{\tau}, s2_o = \sigma \Rightarrow s2_j \xrightarrow{\tau}$. Assume that $s1_i \xrightarrow{\tau} s1_{Aik}$, ($\xrightarrow{\tau}$ is the only kind of transition we have for such states by definition of $\text{Lts}(G)$ in Proposition 3.4). From Proposition 3.4, we have $s1_{Aik} \in f(g1_o \text{ after } \sigma)$. By hypothesis, $G1 \cong G2$, therefore, $\forall \sigma \in \text{Tr}(G1)$, $\text{Ac}1(g1_o \text{ after } \sigma) = \text{Ac}2(g2_o \text{ after } \sigma)$. It follows that there is a state $s2_{Ajl} \in f(g2_o \text{ after } \sigma)$, such that $Aik = Ajl$, and by definition of $\text{Lts}(G)$ in Proposition 3.4, $s2_j \xrightarrow{\tau} s2_{Ajl}$. Therefore, $(s1_{Aik}, s2_{Ajl}) \in R2$. The second part of the proof (assume that $s2_j \xrightarrow{\tau} s2_{Ajl} \dots$) is symmetrical.

- Consider an element $(s1_{Aik}, s2_{Ajl}) \in R2$. It follows that $s1_{Aik} \in f(g1_o \text{ after } \sigma), s2_{Ajl} \in f(g2_o \text{ after } \sigma)$, for some $\sigma \in \text{Tr}(S1)$, and $Aik = Ajl$. Now assume that $s1_{Aik} \xrightarrow{a} s1_l$, (\xrightarrow{a} is the only kind of transition we have for such states by definition of $\text{Lts}(G)$ in Proposition 3.4). By definition of $\text{Lts}(G)$, this is possible if and only if $s1_l \xrightarrow{a} s1_m$. Since $G1 \cong G2$, then we also have $s2_{Ajl} \xrightarrow{a} s2_m$ in $G2$. Since $Aik = Ajl$ and $Aik \in Aik$ it follows that $Ajl \in Aik$. By definition of $\text{Lts}(G)$, we have $s2_{Ajl} \xrightarrow{a} s2_m$. We have $s1_o = \sigma.a \Rightarrow s1_l \xrightarrow{\tau}, s2_o = \sigma.a \Rightarrow s2_m \xrightarrow{\tau}$, for some $\sigma.a \in \text{Tr}(S1)$. Therefore, $(s1_l, s2_m) \in R1$. The second part of the proof (assume that $s2_{Ajl} \xrightarrow{a} s2_m$) is identical. We have proved that R is a bisimulation. Therefore, if $G1 \cong G2$ then $\text{Lts}(G1) \cong \text{Lts}(G2)$. Consequently, $G1 \cong G2$ iff $\text{Lts}(G1) \cong \text{Lts}(G2)$.

2 - From Proposition 3.2 and Lemma 3.1, $S1$ and $S2$ have the same set of cyclic traces, if and only if $G1$ and $G2$ have the set of cyclic traces. From (1), $G1 \cong G2$ iff $\text{Lts}(G1) \cong \text{Lts}(G2)$. Therefore, $G1 \cong G2$ iff $\text{Lts}(G1) \cong \text{Lts}(G2)$.

3 - From (1), we know that $G1 \cong G2$ iff $\text{Lts}(G1) \cong \text{Lts}(G2)$. Due to the correspondence between states of an $G1$ (respectively $G2$) and states of $\text{Lts}(G1)$ (respectively $\text{Lts}(G2)$), it is obvious that there is a bisimulation between $G1$ and $G2$ where each state of $G1$ is related to one and only state of $G2$, if and only if there is a bisimulation between $\text{Lts}(G1)$ and $\text{Lts}(G2)$ where each state of $\text{Lts}(G1)$ is related to one and only state of $\text{Lts}(G2)$.

Proposition 4.1

Consider the AGs $G1 = \langle Sg1, L1, Ac1, Tg1, g1_o \rangle$, $G2 = \langle Sg2, L2, Ac2, Tg2, g2_o \rangle$.
 We have to prove that $\text{Merge}(G1, G2)$ satisfies the consistency constraints $Co, C1, C2, C3$, and $C4$.

For that, we have to prove that $\langle Sg_3, L1 \cup L2, Ac_3, Tg_3, \langle g_1, g_2 \rangle \rangle$ as defined in Definition 4.1 satisfies these requirements:

- Co: By definition of the acceptance sets of the states in Sg_3 , we have $\forall g_i \in Sg_3, \exists B(g_i)$ because G_1 and G_2 are AGs, $\forall g_{1j} \in Sg_1, \exists Ac_1(g_{1j})$ and $\forall g_{2k} \in Sg_2, \exists Ac_2(g_{2k})$.
- C1 and C2: The constraints C1 and C2 are satisfied by definition of the transition function Tg_3 and the fact that G_1 and G_2 are AGs. For each state in Sg_3 , let A in $L3$ and interaction a in A , there is one and only transition labelled by a from this state. For a state g_i in Sg_3 , there is a transition from g_i labelled by interaction a only if $\exists A \in Ac_3(g_i)$ and that $a \in A$.
- C3 (closure under union): $\forall g_i \in Sg_3$, if $g_i = g_{1j}$ or $g_i = g_{2k}$, then $Ac_3(g_i) = Ac_1(g_{1j}) \cup Ac_2(g_{2k})$. If $A_1, A_2 \in Ac_3(g_i)$ then $A_1 \cap A_2 = A_{2k1}$ and $A_1 \cup A_2 = A_{2k2}$, where $A_{2k1} \in Ac_1(g_{1j})$ and $A_{2k2} \in Ac_2(g_{2k})$ by definition of Ac_3 . Since Ac_1 and Ac_2 satisfy C3, we have $(A_{1j1} \cap A_{1j2}) \cup (A_{2k1} \cap A_{2k2}) \in Ac_3(g_i)$ and $(A_{1j1} \cup A_{1j2}) \cap (A_{2k1} \cup A_{2k2}) \in Ac_3(g_i)$ shows that $(A_{1j1} \cap A_{1j2}) \cup (A_{2k1} \cap A_{2k2}) \in Ac_3(g_i)$ and $(A_{1j1} \cup A_{1j2}) \cap (A_{2k1} \cup A_{2k2}) \in Ac_3(g_i)$ in all cases where $g_i = g_{1j}$, or $g_i = g_{2k}$, the proof is obvious since Ac_1 and Ac_2 satisfy C3 by hypothesis. The proof of satisfaction of C4 is similar to the proof for C3.

$\langle Sg_3, L1 \cup L2, Ac_3, Tg_3, \langle g_1, g_2 \rangle \rangle$ is an AG. Consequently, $Merge(G_1, G_2) = \text{reachable}(\langle Sg_3, L1 \cup L2, Ac_3, Tg_3, \langle g_1, g_2 \rangle \rangle)$ is an AG.

Proposition 4.2

Let $G_1 = \langle Sg_1, L1, Ac_1, Tg_1, g_{10} \rangle$, $G_2 = \langle Sg_2, L2, Ac_2, Tg_2, g_{20} \rangle$ and $G_3 = \langle Sg_3, L3, Ac_3, Tg_3, g_{30} \rangle$.

- (a) $Merge(G_1, G_2) =_g Merge(G_2, G_1)$:
 let Sg_4 and Sg_5 be the set of states of $Merge(G_1, G_2)$ and $Merge(G_2, G_1)$, respectively. The relation $\{(\langle g_{1i}, g_{2j} \rangle, \langle g_{2j}, g_{1i} \rangle) : g_{1i} \in Sg_1, g_{2j} \in Sg_2, \langle g_{1i}, g_{2j} \rangle \in Sg_4 \text{ and } \langle g_{2j}, g_{1i} \rangle \in Sg_5\}$ $\cup \{(g_i, g_i') : g_i \in Sg_4, g_i' \in Sg_5, \text{ and } g_i = g_i'\}$ is a bisimulation containing the pair $(\langle g_{10}, g_{20} \rangle, \langle g_{20}, g_{10} \rangle)$ and each state of $Merge(G_1, G_2)$ is related to one and only state of $Merge(G_2, G_1)$ and vice et versa. The AGs G_1 and G_2 have symmetrical roles in the definition of $Merge(G_1, G_2)$.
- (b) $Merge(Merge(G_1, G_2), G_3) =_g Merge(G_1, Merge(G_2, G_3))$:
 let Sg_4 and Sg_5 be the set of states of $Merge(Merge(G_1, G_2), G_3)$ and $Merge(G_1, Merge(G_2, G_3))$, respectively. The relation $\{(\langle \langle g_{1i}, g_{2j} \rangle, g_{3k} \rangle, \langle g_{1i}, \langle g_{2j}, g_{3k} \rangle \rangle) : g_{1i} \in Sg_1, g_{2j} \in Sg_2, g_{3k} \in Sg_3, \langle \langle g_{1i}, g_{2j} \rangle, g_{3k} \rangle \in Sg_4 \text{ and } \langle g_{1i}, \langle g_{2j}, g_{3k} \rangle \rangle \in Sg_5\} \cup \{(g_i, g_i') : g_i \in Sg_4, g_i' \in Sg_5, \text{ and } g_i = g_i'\}$ is a bisimulation containing the pair $(\langle \langle g_{10}, g_{20} \rangle, g_{30} \rangle, \langle g_{10}, \langle g_{20}, g_{30} \rangle \rangle)$ and each state in Sg_4 is related to one and only state of Sg_5 and vice et versa.

Proposition 4.3

Given the AGs $G_1 = \langle Sg_1, L1, Ac_1, Tg_1, g_{10} \rangle$, $G_2 = \langle Sg_2, L2, Ac_2, Tg_2, g_{20} \rangle$,

we have to prove that $\text{Merge}(G1, G2) \text{ ext}_\sigma G1$:

- a - Consider an arbitrary trace σ in $G1$ with $g1_0 = \sigma \Rightarrow g1_i$. From a definition of $\text{Merge}(G1, G2)$, $\exists g2_j \in Sg2$ such that $\langle g1_0, g2_0 \rangle = \sigma \Rightarrow g_i$, where $g_i = g1_i$ or $g_i = \langle g1_i, g2_j \rangle$, for some state $g2_j \in Sg2$. Consequently, $\text{Tr}(G1) \subseteq \text{Tr}(\text{Merge}(G1, G2))$.
- b - From (a) above, if $g1_0 = \sigma \Rightarrow g1_i$, then $\exists g2_j \in Sg2$ such that $\langle g1_0, g2_0 \rangle = \sigma \Rightarrow g_i$, where $g_i = g1_i$ or $g_i = \langle g1_i, g2_j \rangle$, for some state $g2_j \in Sg2$. From the definition of Merge we have $\text{Ac3}(g_i) \subseteq \text{Ac1}(g1_i)$, it follows that $\text{Ac3}(g_i) \subseteq \text{Ac1}(g1_i)$. If $g_i = \langle g1_i, g2_j \rangle$ for some $g2_j \in Sg2$, by definition of Merge we have $\text{Ac3}(g_i) = \{X1 \ X2 \mid X1 \in \text{Ac1}(g1_i) \ \& \ X2 \in \text{Ac2}(g2_j)\}$. It follows that $\text{Ac3}(g_i) \subseteq \text{Ac1}(g1_i)$, since for any $X \in \text{Ac3}(g_i)$ there is an $X1 \in \text{Ac1}(g1_i)$ such that $X1 \subseteq X$. Consequently, $\text{Merge}(G1, G2) \text{ ext}_\sigma G1$.

Proposition 4.4

Let $G1 = \langle Sg1, L1, Ac1, Tg1, g1_0 \rangle$ and $G2 = \langle Sg2, L2, Ac2, Tg2, g2_0 \rangle$.

Consider an elementary cyclic trace $\sigma = a1.a2...an$ in $G1$. It follows that $\exists g1_i, g1_{i+1}, \dots, g1_{i+n-2}$ in $g1$, such that $g1_i \xrightarrow{a1} g1_{i+1}, g1_{i+1} \xrightarrow{a2} g1_{i+2}, \dots, g1_{i+n-2} \xrightarrow{an} g1_0$, with $g1_j \subseteq g1_0$, for $j = i, \dots, i+n-2$.

Sufficient condition:

$\sigma \in \text{Tr}(G2)$, it follows that $\sigma = \sigma'.aj.\sigma''$ and $g2_0 = a1 \Rightarrow g2_k, g2_k = a2 \Rightarrow g2_{k+1}, \dots, g2_{k+j-3} = a_{j-1} \Rightarrow g2_{k+j-2}$, and $g2_{k+j-2} \xrightarrow{aj} g1_{i+j-1}$. From the definition of $\text{Merge}(G1, G2)$, we have $\langle g1_0, g2_0 \rangle = a1 \Rightarrow \langle g1_i, g2_k \rangle, \langle g1_i, g2_k \rangle = a2 \Rightarrow \langle g1_{i+1}, g2_{k+1} \rangle, \dots, \langle g1_{i+j-3}, g2_{k+j-3} \rangle = a_{j-1} \Rightarrow \langle g1_{i+j-2}, g2_{k+j-2} \rangle, \langle g1_{i+j-2}, g2_{k+j-2} \rangle = aj \Rightarrow g1_{i+j-1}, \dots, g1_{i+n-2} = an \Rightarrow \langle g1_0, g2_0 \rangle$ in $\text{Merge}(G1, G2)$, which means that σ is a cyclic trace in $\text{Merge}(G1, G2)$.

σ is a cyclic trace in $G2$, it follows $\exists g2_k, g2_{k+1}, \dots, g2_{k+n-2}$ in $Sg2$ such that $g2_0 = a1 \Rightarrow g2_k, g2_k = a2 \Rightarrow g2_{k+1}, \dots, g2_{k+n-2} = an \Rightarrow g2_0$. From the definition of $\text{Merge}(G1, G2)$, we have $\langle g1_0, g2_0 \rangle = a1 \Rightarrow \langle g1_i, g2_k \rangle, \langle g1_i, g2_k \rangle = a2 \Rightarrow \langle g1_{i+1}, g2_{k+1} \rangle, \dots$, and $\langle g1_{i+n-2}, g2_{k+n-2} \rangle = an \Rightarrow \langle g1_0, g2_0 \rangle$ in $\text{Merge}(G1, G2)$, which means that σ is a cyclic trace in $\text{Merge}(G1, G2)$.

Necessary Condition:

Assume that $\sigma \in \text{Tr}(G2)$ and σ is not a cyclic trace in $G2$. It follows that $\exists g2_k$, such that $g2_0 = \sigma \Rightarrow g2_k$, with $g2_k \not\subseteq g2_0$. By definition of $\text{Merge}(G1, G2)$, we have $\langle g1_0, g2_0 \rangle = \sigma \Rightarrow \langle g1_0, g2_k \rangle$, with $\langle g1_0, g2_k \rangle \not\subseteq \langle g1_0, g2_0 \rangle$. Consequently, σ is not a cyclic in $\text{Merge}(G1, G2)$, which ends the proof that $(\sigma \in \text{Tr}(G2) \text{ or } \sigma \text{ is a cyclic trace in } G2)$ is a necessary condition.

Proposition 4.5

Let $G1 = \langle Sg1, L1, Ac1, Tg1, g1_0 \rangle$ and $G2 = \langle Sg2, L2, Ac2, Tg2, g2_0 \rangle$

- 1 - Equivalence between (a) and (b): we know that $\text{Merge}(G1, G2)$ preserves the cyclic traces of $G1$, iff any elementary cyclic trace in $G1$ is preserved, as cyclic trace, in $\text{Merge}(G1, G2)$. From Proposition 4.4, we know that any elementary cyclic trace σ in $G1$ is a cyclic trace in $\text{Merge}(G1, G2)$, iff σ is a

cyclic trace in G_2 or $\sigma \in \text{Tr}(G_2)$. It follows that $\text{Merge}(G_1, G_2)$ preserves the cyclic traces of G_1 iff any elementary cyclic trace σ in G_1 is a cyclic trace in G_2 or $\sigma \in \text{Tr}(G_2)$.

2 - Equivalence between (b) and (c):

2- 1 - (c) implies (b): obvious since any elementary cyclic trace is a cyclic trace.

2 - 2 - (b) implies (c): assume that any elementary cyclic trace σ in G_1 is a cyclic trace in G_2 or $\sigma \in \text{Tr}(G_2)$ and consider an arbitrary cyclic trace σ in G_1 . Any cyclic trace results from the concatenation of elementary cyclic traces therefore $\sigma = \sigma_1.\sigma_2...\sigma_n$, with σ_i as elementary cyclic trace in G_1 . For $i = 1, \dots, n$, σ_i is an elementary cyclic trace in G_1 , by hypothesis, it follows that σ_i is a cyclic trace in G_2 or $\sigma_i \in \text{Tr}(G_2)$, for $i = 1, \dots, n$. Assume that σ_i is a cyclic trace in G_2 for $i = 1, \dots, n$, it follows that $\sigma = \sigma_1.\sigma_2...\sigma_n$ is a cyclic trace in G_2 (concatenation of cyclic traces is a cyclic trace). Now assume that σ_i , for $i=1, \dots, j-1$, are cyclic traces in G_2 and $\sigma_j \in \text{Tr}(G_2)$ with $j < n$. It follows that $\sigma_1.\sigma_2...\sigma_{j-1}$ is a cyclic trace in G_2 , but $\sigma_1.\sigma_2...\sigma_{j-1}.\sigma_j \in \text{Tr}(G_2)$, which means that $\sigma \in \text{Tr}(G_2)$. Therefore, (b) implies (c).

Consequently, the statements (a), (b) and (c) in Proposition 4.5 are equivalent.

Proposition 4.6

Let $G_1 = \langle \text{Sg}_1, L_1, \text{Ac}_1, \text{Tg}_1, g_{1_0} \rangle$ and $G_2 = \langle \text{Sg}_2, L_2, \text{Ac}_2, \text{Tg}_2, g_{2_0} \rangle$.

Consider $\sigma = a_1.a_2\dots.a_n$, an arbitrary elementary cyclic trace in $\text{Merge}(G_1, G_2)$. By definition of the elementary cyclic trace, we have $\langle g_{1_0}, g_{2_0} \rangle = a_1 \Rightarrow g_{1_1} = a_2 \Rightarrow g_{1_2} \dots g_{1_{n-1}} = a_n \Rightarrow \langle g_{1_0}, g_{2_0} \rangle$ with $g_{ij} \in \langle g_{1_0}, g_{2_0} \rangle$, for $j = 1, \dots, n-1$. From the Definition of Merge, we have the following three cases:

- (a) $g_{ij} = \langle g_{1_{ij}}, g_{2_{ij}} \rangle$, with $\langle g_{1_{ij}}, g_{2_{ij}} \rangle \in \langle g_{1_0}, g_{2_0} \rangle$ for $j = 1, \dots, n-1$, which implies that $g_{1_0} = a_1 \Rightarrow g_{1_{11}} = a_2 \Rightarrow g_{1_{12}} \dots g_{1_{1_{n-1}}} = a_n \Rightarrow g_{1_0}$ and $g_{2_0} = a_1 \Rightarrow g_{2_{11}} = a_2 \Rightarrow g_{2_{12}} \dots g_{2_{1_{n-1}}} = a_n \Rightarrow g_{2_0}$. Therefore, σ is a cyclic trace in G_1 and G_2 .
- (b) $g_{ij} = \langle g_{1_{ij}}, g_{2_{ij}} \rangle$ with $\langle g_{1_{ij}}, g_{2_{ij}} \rangle \in \langle g_{1_0}, g_{2_0} \rangle$, for $j = 1, \dots, k$, (for a certain k) and $g_{ij} = g_{1_{ij}} \in \langle g_{1_0} \rangle$, for $j = k+1, \dots, n-1$, which means that $g_{1_0} = a_1 \Rightarrow g_{1_{11}} = a_2 \Rightarrow g_{1_{12}} \dots g_{1_{1_{n-1}}} = a_n \Rightarrow g_{1_0}$. Therefore, σ is a cyclic trace in G_1 .
- (c) $g_{ij} = \langle g_{1_{ij}}, g_{2_{ij}} \rangle$ with $\langle g_{1_{ij}}, g_{2_{ij}} \rangle \in \langle g_{1_0}, g_{2_0} \rangle$, for $j = 1, \dots, k$, (for a certain k) and $g_{ij} = g_{2_{ij}} \in \langle g_{2_0} \rangle$, for $j = k+1, \dots, n-1$, which means that $g_{2_0} = a_1 \Rightarrow g_{2_{11}} = a_2 \Rightarrow g_{2_{12}} \dots g_{2_{1_{n-1}}} = a_n \Rightarrow g_{2_0}$. Therefore σ is a cyclic trace in G_2 .

Consequently, σ is a cyclic trace in G_1 or G_2 .

Proposition 4.7

Let $G_1 = \langle \text{Sg}_1, L_1, \text{Ac}_1, \text{Tg}_1, g_{1_0} \rangle$ and $G_2 = \langle \text{Sg}_2, L_2, \text{Ac}_2, \text{Tg}_2, g_{2_0} \rangle$.

(a) σ is a cyclic in $\text{Merge}(G1, G2)$: $\sigma = \sigma_1.\sigma_2...\sigma_n.\sigma_{n+1}$, with σ_i as elementary cyclic trace in $\text{Merge}(G1, G2)$, for $i=1, \dots, n+1$, for a certain integer n . From Proposition 4.6, σ_i as a cyclic trace in $G1$ or $G2$, for $i=1, \dots, n+1$. Therefore, σ_i is a cyclic trace in $G1$ or $G2$, for $i=1, \dots, n$, and $(\sigma_{n+1} \in \text{Tr}(G1))$ or $\sigma_{n+1} \in \text{Tr}(G2)$.

(b) σ is a noncyclic in $\text{Merge}(G1, G2)$: $\sigma = \sigma'.a_1.a_2...a_m$ with $\langle g1_o, g2_o \rangle = \sigma' \Rightarrow g1_o, g2_o = a_1 \Rightarrow g_{i1} = a_2 \Rightarrow g_{i2} \dots g_{im-1} = a_n \Rightarrow g_{im}$ with $g_{ij} \langle g1_o, g2_o \rangle$, for $j = 1, \dots, m$. σ' is a cyclic trace in $\text{Merge}(G1, G2)$. Therefore, $\sigma' = \sigma'_1.\sigma'_2...\sigma'_n$, with σ'_i as elementary cyclic trace in $\text{Merge}(G1, G2)$, for $i=1, \dots, n$, for a certain integer n . From Proposition 4.6, σ'_i as a cyclic trace in $G1$ or $G2$, for $i=1, \dots, n$.

We have $\langle g1_o, g2_o \rangle = a_1 \Rightarrow g_{i1} = a_2 \Rightarrow g_{i2} \dots g_{im-1} = a_n \Rightarrow g_{im}$ with $g_{ij} \langle g1_o, g2_o \rangle$, for $j = 1, \dots, m$. From the definition of Merge, we have the following three cases:

- (a) $g_{ij} = \langle g1_{ij}, g2_{ij} \rangle$, with $\langle g1_{ij}, g2_{ij} \rangle \langle g1_o, g2_o \rangle$ for $j = 1, \dots, m$, which means that $g1_o = a_1 \Rightarrow g1_{i1} = a_2 \Rightarrow g1_{i2} \dots g1_{im-1} = a_m \Rightarrow g1_m$ and $g2_o = a_1 \Rightarrow g2_{i1} = a_2 \Rightarrow g2_{i2} \dots g2_{im-1} = a_m \Rightarrow g2_m$. Therefore, $a_1.a_2...a_m \in \text{Tr}(G1)$ and $a_1.a_2...a_m \in \text{Tr}(G2)$.
- (b) $g_{ij} = \langle g1_{ij}, g2_{ij} \rangle$ with $\langle g1_{ij}, g2_{ij} \rangle \langle g1_o, g2_o \rangle$, for $j = 1, \dots, k$, (for a certain k) and $g_{ij} = g1_{ij} \langle g1_o \rangle$, for $j = k+1, \dots, n-1$, which means that $g1_o = a_1 \Rightarrow g1_{i1} = a_2 \Rightarrow g1_{i2} \dots g1_{im-1} = a_m \Rightarrow g1_m$. Therefore, $a_1.a_2...a_m \in \text{Tr}(G1)$.
- (c) $g_{ij} = \langle g1_{ij}, g2_{ij} \rangle$ with $\langle g1_{ij}, g2_{ij} \rangle \langle g1_o, g2_o \rangle$, for $j = 1, \dots, k$, (for a certain k) and $g_{ij} = g2_{ij} \langle g2_o \rangle$, for $j = k+1, \dots, n-1$, which means that $g2_o = a_1 \Rightarrow g2_{i1} = a_2 \Rightarrow g2_{i2} \dots g2_{im-1} = a_m \Rightarrow g2_m$. Therefore, $a_1.a_2...a_m \in \text{Tr}(G2)$.

Consequently, any trace σ of $\text{Merge}(G1, G2)$ may be written as $\sigma = \sigma_1.\sigma_2...\sigma_n.\sigma_{n+1}$, with σ_i as a cyclic trace in $G1$ or $G2$, for $i=1, \dots, n$, and $(\sigma_{n+1} \in \text{Tr}(G1))$ or $\sigma_{n+1} \in \text{Tr}(G2)$.

Theorem 4.1

Let $G1 = \langle Sg1, L1, Ac1, Tg1, g1_o \rangle$ and $G2 = \langle Sg2, L2, Ac2, Tg2, g2_o \rangle$

From Proposition 4.3, we have $\text{Merge}(G1, G2) \text{ ext}_g G1, G2$.

From Proposition 4.5, $\text{Merge}(G1, G2)$ preserves the cyclic traces of $G1$ and $G2$.

any cyclic trace σ in $G1$ is a cyclic trace in $G2$ or $\sigma \in \text{Tr}(G2)$.

It follows that $\text{Merge}(G1, G2)$ is a cyclic extension of $G1$ and $G2$, in

any cyclic trace σ in $G1$ is a cyclic trace in $G2$ or $\sigma \in \text{Tr}(G2)$, and reciprocally.

Now, we have to prove that $\text{Merge}(G1, G2)$ is the least common cyclic extension of $G1$ and $G2$. For that, we consider an arbitrary AG $G4 = \langle Sg4, L4, Ac4, Tg4, g4_o \rangle$ such that $G4 \text{ ext}_g G1, G4 \text{ ext}_g G2$ and we will prove that $G4 \text{ ext}_g \text{Merge}(G1, G2)$.

First, we have to prove that any cyclic trace in $\text{Merge}(G1, G2)$ is a cyclic trace in $G4$. Consider a cyclic trace σ in $\text{Merge}(G1, G2)$. $\sigma = \sigma_1.\sigma_2...\sigma_n$ with $\sigma_1, \sigma_2, \dots, \sigma_n$ as elementary cyclic traces in $\text{Merge}(G1, G2)$.

By Proposition 4.6, it follows that σ_i is a cyclic trace in G_1 or G_2 , for $i = 1, \dots, n$. We have σ_i as a cyclic trace in G_1 or G_2 , for $i = 1, \dots, n$. It follows that σ_i is a cyclic trace in G_4 , for $i = 1, \dots, n$, since G_4 is a cyclic extension of G_1 and G_2 . Consequently, σ is a cyclic trace in G_4 (concatenation of cyclic traces is a cyclic trace)

Secondly, we have to prove that $G_4 \text{ ext}_G \text{ Merge}(G_1, G_2)$:

(1) Consider an arbitrary trace σ in $\text{Merge}(G_1, G_2)$. The trace σ can be written as $\sigma = \sigma_1.\sigma_2\dots\sigma_{n-1}.\sigma_n$ with σ_i as cyclic trace in G_1 or G_2 , for $i = 1, \dots, n-1$, and $\sigma_n \in \text{Tr}(G_1)$ or $\sigma_n \in \text{Tr}(G_2)$. $G_4 \text{ ext}_G G_1$ and $G_4 \text{ ext}_G G_2$, it follows that any trace of G_1 (respectively G_2) is a trace of G_4 , and any cyclic trace in G_1 (respectively G_2) is a cyclic trace in G_4 , it follows that σ_i is a cyclic trace in G_4 , for $i = 1, \dots, n-1$, and $\sigma_n \in \text{Tr}(G_4)$. We deduce that $\sigma = \sigma_1.\sigma_2\dots\sigma_{n-1}.\sigma_n \in \text{Tr}(G_4)$.

(2) Consider an arbitrary trace σ in $\text{Merge}(G_1, G_2)$: as previously, the trace σ can be written as $\sigma = \sigma_1.\sigma_2\dots\sigma_{n-1}.\sigma_n$ with σ_i as cyclic trace in G_1 or G_2 , for $i = 1, \dots, n-1$, and $\sigma_n \in \text{Tr}(G_1)$ or $\sigma_n \in \text{Tr}(G_2)$. We have deduced that σ_i is a cyclic trace in G_4 , for $i = 1, \dots, n-1$, and $\sigma_n \in \text{Tr}(G_4)$. $\sigma \in \text{Tr}(\text{Merge}(G_1, G_2))$, it follows that $\exists g_i$ in $\text{Merge}(G_1, G_2)$ such that $\sigma = \langle g_1, g_2 \rangle = \sigma_n \Rightarrow g_i$. Since $\sigma_1, \dots, \sigma_{n-1}$ are (elementary) cyclic traces in $\text{Merge}(G_1, G_2)$, it follows that $\langle g_1, g_2 \rangle = \sigma_n \Rightarrow g_i$. So reasoning for G_4 , $\exists g_1$ in G_1 such that $g_4 = \sigma_n \Rightarrow g_1$ and $g_4 = \sigma_n \Rightarrow g_2$ with $\sigma_n \in \text{Tr}(G_1)$ and $\sigma_n \in \text{Tr}(G_2)$. We deduce that $\exists g_1$ in G_1 such that $g_1 = \sigma_n \Rightarrow g_1$, and by definition of Merge , $g_1 = \langle g_1, g_2 \rangle$ and $\text{Ac}(g_1) = \text{Ac}_1(g_1)$. We have $G_4 \text{ ext}_G G_1$, it follows that $\text{Ac}_4(g_4) = \text{Ac}_1(g_1) = \text{Ac}_1(g_1)$. Reciprocally, if $\sigma_n \in \text{Tr}(G_2)$ and $\sigma_n \in \text{Tr}(G_1)$. If $\sigma_n \in \text{Tr}(G_1)$ and $\sigma_n \in \text{Tr}(G_2)$, $\exists g_1$ in G_1 and $\exists g_2$ in G_2 such that $g_1 = \sigma_n \Rightarrow g_1$, and $g_2 = \sigma_n \Rightarrow g_2$, and by definition of Merge , $g_1 = \langle g_1, g_2 \rangle$ and $\text{Ac}_3(g_1) = \{X_1 \mid X_2 \mid X_1 \in \text{Ac}_1(g_1) \text{ and } X_2 \in \text{Ac}_2(g_2)\}$. We have $G_4 \text{ ext}_G G_1$ and $G_4 \text{ ext}_G G_2$, it follows that $\text{Ac}_4(g_4) = \text{Ac}_3(g_1) = \text{Ac}_1(g_1) \cup \text{Ac}_2(g_2)$. It follows that $\text{Ac}_4(g_4) = \text{Ac}_3(g_1)$, which ends the second part of the proof $G_4 \text{ ext}_G \text{ Merge}(G_1, G_2)$.

Consequently, $G_4 \text{ ext}_G \text{ Merge}(G_1, G_2)$ and $\text{Merge}(G_1, G_2)$ (an arbitrary elementary cyclic trace in $\text{Merge}(G_1, G_2)$).

By Proposition 4.6, it follows that σ_i is a cyclic trace in G_1 or G_2 , for $i = 1, \dots, n$. We have σ_i as a cyclic trace in G_1 or G_2 , for $i = 1, \dots, n$. It follows that σ_i is a cyclic trace in G_4 , for $i = 1, \dots, n$, since G_4 is a cyclic extension of G_1 and G_2 . Consequently, σ is a cyclic trace in G_4 (concatenation of cyclic traces is a cyclic trace)

(a) $\langle g_1, g_2 \rangle = \langle g_1, g_2 \rangle$, with $g_1 \in G_1$ and $g_2 \in G_2$.

$ij, g_{2ij} \rangle \langle g_{1o}, g_{2o} \rangle$, for $j = 1, \dots, n-1$, it follows that $\langle g_{3o}, g_{2o} \rangle = a_1 f g_{3i1}, g_{2i1} \rangle = a_2 f g_{3i2}, g_{2i2} \rangle \dots \langle g_{3in-1}, g_{2in-1} \rangle = a_n f \langle g_{3o}, g_{2o} \rangle$ in $\text{Merge}(G_3, G_2)$ with $\langle g_{3ij}, g_{2ij} \rangle \langle g_{1o}, g_{2o} \rangle$ for $j = 1, \dots, n-1$, since an arbitrary elementary cyclic trace in $\text{Merge}(G_1, G_2)$. By definition of an elementary cyclic trace, we have $\langle g_{1o}, g_{2o} \rangle = a_1 \Rightarrow g_{4i1} = a_2 \Rightarrow g_{4i2} \dots g_{4in-1} = a_n \Rightarrow \langle g_{1o}, g_{2o} \rangle$ with $g_{4ij} \langle g_{1o}, g_{2o} \rangle$, for $j = 1, \dots, n-1$. From the Definition of Merge, we have the following three cases:

- (a) $g_{4ij} = \langle g_{1ij}, g_{2ij} \rangle$, with $\langle g_{1ij}, g_{2ij} \rangle \langle g_{1o}, g_{2o} \rangle$, for $j = 1, \dots, n-1$, it follows that $\langle g_{3o}, g_{2o} \rangle = a_1 \Rightarrow g_{3i1}, g_{2i1} \rangle = a_2 \Rightarrow g_{3i2}, g_{2i2} \rangle \dots \langle g_{3in-1}, g_{2in-1} \rangle = a_n \Rightarrow \langle g_{3o}, g_{2o} \rangle$ in $\text{Merge}(G_3, G_2)$ with $\langle g_{3ij}, g_{2ij} \rangle \langle g_{1o}, g_{2o} \rangle$ for $j = 1, \dots, n-1$, since $g_{3ij} = g_{3o}$ iff $g_{1ij} = g_{1o}$, for $j = 1, \dots, n-1$ (G_1 and G_3 have the same cyclic traces). Therefore, σ is an elementary cyclic in $\text{Merge}(G_3, G_2)$.
- (b) $g_{4ij} = \langle g_{1ij}, g_{2ij} \rangle$ with $\langle g_{1ij}, g_{2ij} \rangle \langle g_{1o}, g_{2o} \rangle$, for $j = 1, \dots, k$, (for a certain k) and $g_{4ij} = g_{1ij} \langle g_{1o}, g_{2o} \rangle$, for $j = k+1, \dots, n-1$, it follows that $\langle g_{3o}, g_{2o} \rangle = a_1 \Rightarrow \langle g_{3i1}, g_{2i1} \rangle \dots \langle g_{3ik-1}, g_{2ik-1} \rangle = a_k \Rightarrow \langle g_{3ik}, g_{2ik} \rangle = a_{k+1} \Rightarrow g_{3ik+1} \dots g_{3in-1} = a_n \Rightarrow \langle g_{3o}, g_{2o} \rangle$ in $\text{Merge}(G_3, G_2)$ with $\langle g_{3ij}, g_{2ij} \rangle \langle g_{3o}, g_{2o} \rangle$ for $j = 1, \dots, k$, and $g_{3ij} = g_{3o}$, for $j = k+1, \dots, n-1$, since $g_{3ij} = g_{3o}$ iff $g_{1ij} = g_{1o}$, for $j = 1, \dots, n-1$ (G_1 and G_3 have the same cyclic traces)). Therefore, $\langle g_{3o}, g_{2o} \rangle = a_1 \Rightarrow g_{5i1} = a_2 \Rightarrow g_{5i2} \dots g_{5in-1} = a_n \Rightarrow \langle g_{3o}, g_{2o} \rangle$ with $g_{5ij} \langle g_{1o}, g_{2o} \rangle$, for $j = 1, \dots, n-1$, which means that σ is an elementary cyclic in $\text{Merge}(G_3, G_2)$.
- (c) $g_{4ij} = \langle g_{1ij}, g_{2ij} \rangle$ with $\langle g_{1ij}, g_{2ij} \rangle \langle g_{1o}, g_{2o} \rangle$, for $j = 1, \dots, k$, (for a certain k) and $g_{4ij} = g_{2ij} \langle g_{1o}, g_{2o} \rangle$, for $j = k+1, \dots, n-1$, it follows that $\langle g_{3o}, g_{2o} \rangle = a_1 \Rightarrow \langle g_{3i1}, g_{2i1} \rangle \dots \langle g_{3ik-1}, g_{2ik-1} \rangle = a_k \Rightarrow \langle g_{3ik}, g_{2ik} \rangle = a_{k+1} \Rightarrow g_{2ik+1} \dots g_{2in-1} = a_n \Rightarrow \langle g_{3o}, g_{2o} \rangle$ in $\text{Merge}(G_3, G_2)$ with $\langle g_{3ij}, g_{2ij} \rangle \langle g_{3o}, g_{2o} \rangle$ for $j = 1, \dots, k$, since $g_{3ij} = g_{3o}$ iff $g_{1ij} = g_{1o}$ for $j = 1, \dots, k$ (G_1 and G_3 have the same cyclic traces)) and $g_{2ij} = g_{2o}$, for $j = k+1, \dots, n-1$. Therefore, $\langle g_{3o}, g_{2o} \rangle = a_1 \Rightarrow g_{5i1} = a_2 \Rightarrow g_{5i2} \dots g_{5in-1} = a_n \Rightarrow \langle g_{3o}, g_{2o} \rangle$ with $g_{5ij} \langle g_{1o}, g_{2o} \rangle$, for $j = 1, \dots, n-1$, which means that σ is an elementary cyclic in $\text{Merge}(G_3, G_2)$.

The proof for any elementary cyclic trace in $\text{Merge}(G_3, G_2)$ is an elementary cyclic trace in $\text{Merge}(G_1, G_2)$ is symmetrical. Consequently, $\text{Merge}(G_1, G_2)$ and $\text{Merge}(G_3, G_2)$ have the same set of (elementary) cyclic traces.

2 - $\text{Merge}(G_1, G_2) \cong \text{Merge}(G_3, G_2)$:

2 - 1 - $\text{Tr}(\text{Merge}(G_1, G_2)) = \text{Tr}(\text{Merge}(G_3, G_2))$:

Consider a trace $\sigma \in \text{Tr}(\text{Merge}(G_1, G_2))$. $\sigma = \sigma_1 \sigma_2 \dots \sigma_n \sigma_{n+1}$, with σ_i as elementary cyclic trace in $\text{Merge}(G_1, G_2)$, for $i = 1, \dots, n$, and ($\sigma_{n+1} \in \text{Tr}(G_1)$ or $\sigma_{n+1} \in \text{Tr}(G_2)$). It follows, from (1) above, that σ_i is an elementary cyclic trace in $\text{Merge}(G_3, G_2)$, for $i = 1, \dots, n$. $\text{Merge}(G_3, G_2) \text{ ext}_g G_3$ and G_2 and $G_1 \text{ c}_g G_3$, we deduce that ($\sigma_{n+1} \in \text{Tr}(G_3)$ or $\sigma_{n+1} \in \text{Tr}(G_2)$). Therefore, $\sigma = \sigma_1 \sigma_2 \dots \sigma_n \sigma_{n+1} \in$

$\text{Tr}(\text{Merge}(G3, G2))$. The proof for any trace σ of $\text{Merge}(G3, G2)$ is a trace of $\text{Merge}(G3, G2)$ is symmetrical.

2 - 2 - $\forall \sigma \in \text{Tr}(\text{Merge}(G1, G2)), \text{Ac4}(\langle g1_0, g2_0 \rangle \text{ after } \sigma) = \text{Ac5}(\langle g3_0, g2_0 \rangle \text{ after } \sigma)$:

Consider a trace $\sigma \in \text{Tr}(\text{Merge}(G1, G2))$. $\sigma = \sigma_1.\sigma_2 \dots \sigma_n.\sigma_{n+1}$ where σ_i as elementary cyclic trace in $\text{Merge}(G1, G2)$ and $\text{Merge}(G3, G2)$, for $i = 1, \dots, n$, and $(\sigma_{n+1} \in \text{Tr}(G1) \text{ (and } \sigma_{n+1} \in \text{Tr}(G3) \text{)})$ or $\sigma_{n+1} \in \text{Tr}(G2)$. Therefore, $\langle g1_0, g2_0 \rangle \text{ after } \sigma = \langle g1_0, g2_0 \rangle \text{ after } \sigma_{n+1}$ and $\langle g3_0, g2_0 \rangle \text{ after } \sigma = \langle g3_0, g2_0 \rangle \text{ after } \sigma_{n+1}$. $\text{Ac4}(\langle g1_0, g2_0 \rangle \text{ after } \sigma) = \text{Ac5}(\langle g1_0, g2_0 \rangle \text{ after } \sigma)$, iff $\text{Ac4}(\langle g1_0, g2_0 \rangle \text{ after } \sigma_{n+1}) = \text{Ac5}(\langle g3_0, g2_0 \rangle \text{ after } \sigma_{n+1})$. We have three cases:

- $\sigma_{n+1} \in \text{Tr}(G1)$ ($\sigma_{n+1} \in \text{Tr}(G3)$): $\text{Ac4}(\langle g1_0, g2_0 \rangle \text{ after } \sigma_{n+1}) = \text{Ac1}(g1_0 \text{ after } \sigma_{n+1}) = \text{Ac3}(g3_0 \text{ after } \sigma_{n+1}) = \text{Ac5}(\langle g3_0, g2_0 \rangle \text{ after } \sigma_{n+1})$, since $G1 \text{ } c_g \text{ } G3$.
- $\sigma_{n+1} \in \text{Tr}(G2)$: $\text{Ac4}(\langle g1_0, g2_0 \rangle \text{ after } \sigma_{n+1}) = \text{Ac2}(g2_0 \text{ after } \sigma_{n+1}) = \text{Ac5}(\langle g3_0, g2_0 \rangle \text{ after } \sigma_{n+1})$.
- $\sigma_{n+1} \in \text{Tr}(G1)$ ($\sigma_{n+1} \in \text{Tr}(G3)$) and $\sigma_{n+1} \in \text{Tr}(G2)$: $\text{Ac4}(\langle g1_0, g2_0 \rangle \text{ after } \sigma_{n+1}) = \{X1 \text{ } X2 \mid X1 \in \text{Ac1}(g1_0 \text{ after } \sigma_{n+1}) \text{ and } X2 \in \text{Ac2}(g2_0 \text{ after } \sigma_{n+1})\}$ and $\text{Ac5}(\langle g3_0, g2_0 \rangle \text{ after } \sigma_{n+1}) = \{X3 \text{ } X2 \mid X3 \in \text{Ac3}(g3_0 \text{ after } \sigma_{n+1}) \text{ and } X2 \in \text{Ac2}(g2_0 \text{ after } \sigma_{n+1})\}$. Since $G1 \text{ } c_g \text{ } G3$, $\text{Ac1}(g1_0 \text{ after } \sigma_{n+1}) = \text{Ac3}(g3_0 \text{ after } \sigma_{n+1})$. It follows that $\text{Ac4}(\langle g1_0, g2_0 \rangle \text{ after } \sigma_{n+1}) = \text{Ac5}(\langle g3_0, g2_0 \rangle \text{ after } \sigma_{n+1})$.

$\text{Merge}(G1, G2) \text{ } c_g \text{ } \text{Merge}(G3, G2)$ and a trace σ is cyclic in $\text{Merge}(G1, G2)$ iff σ is cyclic in $\text{Merge}(G3, G2)$.
Consequently, $\text{Merge}(G1, G2) \text{ } c_g \text{ } \text{Merge}(G3, G2)$.