

Action Refinement for Probabilistic Processes with True Concurrency Models

Harald Fecher, Mila Majster-Cederbaum, and Jinzhao Wu

Universität Mannheim
Fakultät für Mathematik und Informatik D7, 27
68131 Mannheim, Germany
{hfecher,mcb,wu}@pi2.informatik.uni-mannheim.de

Abstract. In this paper, we develop techniques of action refinement for probabilistic processes within the context of a probabilistic process algebra. A semantic counterpart is carried out in a non-interleaving causality based setting, probabilistic bundle event structures. We show that our refinement notion has the following nice properties: the behaviour of the refined system can be inferred compositionally from the behaviour of the original system and from the behaviour of the systems substituted for actions; the probabilistic extensions of pomset trace equivalence and history preserving bisimulation equivalence are both congruences under the refinement; and with respect to a cpo-based denotational semantics the syntactic and semantic refinements coincide with each other up to the aforementioned equivalence relations when the internal actions are abstracted away.

1 Introduction

Process algebras are a frequently used tool for the specification and verification of concurrent systems. In process algebras, actions are used to denote the basic entities of systems. By an action we understand here any activity which is considered as a conceptual entity on a chosen level of abstraction. This allows the representation of systems in a hierarchical way, i.e. actions on a higher level are interpreted by more complicated processes on a lower level. Such a change in the level of abstraction is usually referred to as *action refinement*, which is a core operation in the methodology of top-down system design.

Action refinement for classical concurrent systems without probabilistic constraints has been thoroughly discussed in the literature [2,13]. The models of concurrency can roughly be distinguished in two kinds: interleaving based models, in which the independent execution of two processes is modelled by specifying the possible interleaving of their actions; and true concurrency models, in which the causal relations between actions are represented explicitly.

Without further restriction or relaxation on actions or refinement operations, most of the equivalence relations are not preserved under refinement in the interleaving approach [10]. These equivalence relations, however, are often used

to establish the correctness of the implementation with respect to the specification of concurrent systems. This problem can be overcome by moving to true concurrency models. Also, in the system design phase the local causal dependencies between actions are important. Interleaving with actions of other parts of the system burdens the design. Partial order models are considered to be much more appropriate here. Moreover, a causality based model does not suffer from the state explosion problem. Parallelism leads to the sum of the components, rather than to their product as in the interleaving approach.

We study action refinement for truly concurrent systems with probabilistic constraints. Probabilistic phenomena are important in many areas of computing. Therefore, formal tools for describing and reasoning about such systems are needed. A number of probabilistic process calculi have been proposed [11,12]. Our aim is to achieve higher reliability with respect not only to the functional correctness but also to the performance issue. In our setting, probabilistic concurrent systems are described in a probabilistic LOTOS-like process algebra [3] proposed in [6,15]. It is actually a synthesis of CCS [23] and CSP [14]. We use probabilistic bundle event structures [15,18] as the semantic model. Note that bundle event structures have been shown to adequately deal with e.g. parallel composition, and the method to attach probabilistic information does not complicate the theory and it is simple and practical. We are unaware of any other work on action refinement in a partial-order setting associated with probabilistic information.

Current probabilistic process algebras usually use probabilistic extensions of labelled transition systems as an underlying semantic model. It is quite common to distinguish between probabilistic and nonprobabilistic transitions in these models. The main problem with this approach is the intertwining of these types of transitions. This is to say, it is not clear what the intended meaning of a probability attached to a transition is in the presence of a competitive nonprobabilistic transition. Such situations can happen if e.g. combinations of parallel composition and probabilistic choice are taken. Several solutions have been proposed for this problem, but most of them lose the property of backward compatibility with the nonprobabilistic semantics. A causality-based model does not have such problems [7,15].

There are essentially two interpretations of action refinement, called syntactic and semantic. Syntactic action refinement, where actions occurring in a process are replaced by more complicated ones, yields a more detailed process description [2,13]. Due to its definitional clarity, syntactic action refinement can be easily used without too much insight on the semantics. Semantic action refinement is carried out in the semantic domain. It avoids a confusion of the abstraction levels, which can happen in syntactic refinement. Such a confusion may result in undesirable situations [10,13].

We adopt the methodology that refinement is modelled as an operator. After defining syntactic and semantic action refinement in the probabilistic process algebra, we discuss three common issues of interest: *safety*, *congruence* and *coincidence* problems.

A refinement operation should be safe. That is, the behaviour of the refined system should be the refinement of the behaviour of the original system (in some sense), and vice versa. Equivalence relations are often used to capture when two systems are considered to exhibit the same behaviour. To examine the well-definedness of the refinement operator, we have to try to find such equivalence relations which are congruences under it. The result that syntactic and semantic refinements coincide can give a clear understanding of the concept of action refinement, and has important applications in system verifications [20,21]: the refined syntactic specification can be modelled by semantic action refinement, and the refined semantic model can be specified by syntactic action refinement.

The main contributions of this paper are:

- the notions of syntactic and semantic refinement operators in the probabilistic process algebra;
- the verification of safety of refinement;
- the congruence results about pomset trace and history preserving bisimulation equivalences; and
- the coincidence result of semantic refinement with syntactic refinement.

This paper is a further development of [10], where action refinement in event structures without probabilistic information is investigated.

2 A Probabilistic Process Algebra

In this section, we fix the process algebraic framework that is used to develop probabilistic concurrent systems, and we describe a cpo-based denotational semantics.

2.1 Syntax

Assume a given set Θ of observable actions and an invisible internal action τ ($\tau \notin \Theta$). Action \surd ($\surd \notin \Theta \cup \{\tau\}$) indicates the successful termination of a process. Let $\Omega = \Theta \cup \{\tau, \surd\}$.

Definition 2.1.1 (*Probabilistic Formalism L*).

$$B ::= 0 \mid 1 \mid a.B \mid B; B \mid B \parallel_A B \mid B + B \mid B +_p B \mid B \setminus A \mid B[\lambda] \mid x \mid \mu x.B.$$

Here, $a \in \Theta \cup \{\tau\}$, $A \subseteq \Theta$, $\lambda: \Omega \rightarrow \Omega$ is a relabelling function such that $\lambda(\tau) = \tau$, $\lambda(\surd) = \surd$ and $\lambda(a) \neq \surd$ for $a \in \Theta$, $x \in V$, V is a set of process variables, and $p \in (0, 1)$, i.e. $0 < p < 1$.

We assume $\text{dom}(\lambda) = \{a \in \Omega \mid \lambda(a) \neq a\}$, representing the set of actions that are really relabelled by λ . The operators have the following intuitive meaning:

0 denotes inaction. 1 represents the process that terminates successfully.

$a.B_1$ denotes the prefix of action a before process B_1 . $B_1[\lambda]$ denotes the relabelling of B_1 according to λ . $B_1 \setminus A$ behaves as B_1 , except that the actions in A are abstracted, i.e., turned into τ -actions.

$B_1; B_2$ denotes the sequential composition of processes B_1 and B_2 , where the control is passed to B_2 by the successful termination of B_1 . $B_1 \parallel_A B_2$ denotes the parallel composition of B_1 and B_2 , where B_1 and B_2 must perform any actions in $A \cup \{\surd\}$ simultaneously, while the other actions are executed independently from each other.

We distinguish between a non-deterministic and a probabilistic choice. $B_1 + B_2$ indicates the non-deterministic choice between the behaviours described by processes B_1 and B_2 . $B_1 +_p B_2$ behaves like B_1 with probability p or like B_2 with probability $1 - p$. This distinction is important. From a design perspective it is necessary to express choices for which the probability of an alternative is left unspecified. Such quantitative knowledge may either be absent at the current stage of design or it may be deliberately left unspecified. Therefore, one should not be forced to associate such quantity with an alternative. When going from an abstract specification to a more concrete specification, it is useful to consider the replacement of some non-deterministic choices by probabilistic choices.

Recursive behaviour is described by $\mu x.B_1$. It can be understood through $x := B_1$, where x may occur in the body B_1 .

The probabilistic choice between processes B_1 and B_2 should not be influenced by the environment. So we have to add some constraints on B_1 and B_2 . These constraints guarantee that $B_1 +_p B_2$ induces an independent stochastic experiment.

We first introduce two subsidiary predicates pc and ppc .

Let $ppc : L \rightarrow \{true, false\}$ be defined as follows:

$$\begin{aligned} ppc(B_1 +_p B_2) &= (ppc(B_1) \vee B_1 = \tau.B'_1) \wedge (ppc(B_2) \vee B_2 = \tau.B'_2), \\ ppc(B_1; B_2) &= ppc(B_1), \\ ppc(\circ B_1) &= ppc(B_1) \text{ for } \circ \in \{\setminus A, [\lambda]\}, \\ ppc(\mu x.B_1) &= ppc(B_1), \\ ppc &\text{ is } false \text{ for all other syntactical constructs.} \end{aligned}$$

Let $pc : L \rightarrow \{true, false\}$ be defined as follows:

$$\begin{aligned} pc(B_1 +_p B_2) &= true, \\ pc(B_1; B_2) &= pc(B_1), \\ pc(B_1 \parallel_A B_2) &= pc(B_1) \vee pc(B_2), \\ pc(\circ B_1) &= pc(B_1) \text{ for } \circ \in \{\setminus A, [\lambda]\}, \\ pc(\mu x.B_1) &= pc(B_1), \\ pc &\text{ is } false \text{ for all other syntactical constructs.} \end{aligned}$$

In fact, $ppc(B)$ holds if a pure probabilistic choice has to happen next in B . $pc(B)$ holds if a probabilistic choice occurs in a proper component of B .

Definition 2.1.2 (*Probabilistic Process Algebra PPA*). $PPA = \{B \in L \mid ppa(B)\}$, where $ppa : L \rightarrow \{true, false\}$ is defined as:

$$\begin{aligned} ppa(0) &= true, \\ ppa(1) &= true, \end{aligned}$$

$$\begin{aligned}
ppa(x) &= true \text{ for } x \in V, \\
ppa(\circ B_1) &= ppa(B_1) \text{ for } \circ \in \{a., \setminus A, [\lambda]\}, \\
ppa(B_1 \circ B_2) &= ppa(B_1) \wedge ppa(B_2) \text{ for } \circ \in \{;, \|_A\}, \\
ppa(B_1 + B_2) &= \neg pc(B_1) \wedge \neg pc(B_2) \wedge ppa(B_1) \wedge ppa(B_2), \\
ppa(B_1 +_p B_2) &= ppc(B_1 +_p B_2) \wedge ppa(B_1) \wedge ppa(B_2), \\
ppa(\mu x. B_1) &= ppa(B_1).
\end{aligned}$$

Hereafter, the elements of PPA , denoted as B and B_k , are called (*probabilistic*) *expressions*. The alphabet of expression B , namely the set of all observable actions occurring in B , is denoted by A_B .

Example 2.1.3. $B' = \tau.a.1 + (\tau.b.0 +_{0.5} \tau.c.1)$ is not an expression, whereas the following B and B_c are expressions:
 $B = r.((\tau.a.s.0 +_{0.4} \tau.b.s.0) +_{0.7} \tau.c.s.0)$
 $B_c = \tau.c_1.1 +_{0.8} \tau.c_2.1$

2.2 Semantic Model

We use probabilistic bundle event structures as the semantic model of the probabilistic process algebra.

Probabilistic Event Structures. A bundle event structure [18] consists of events labelled with actions and two relations between events for causality and for conflict. When an event occurs, the action labelled to it occurs. The symmetric conflict relation, denoted \sharp , is a binary relation between events, and the intended meaning of $e\sharp e'$ is that events e and e' cannot both occur in a single system run. Causality is represented by a binary bundle relation, denoted \mapsto . It relates a set of events, where all events have to be pairwise in conflict, and an event. $X \mapsto e$ means that if event e happens in a system run, exactly one event in event set X has happened before and caused e . X is called a *bundle-set*.

The basic idea of probabilistic event structures is to incorporate fixed probabilities in event structures by associating probabilities with some events [15]. Suppose we have an event e and we decorate this event with probability $p \in (0, 1)$. The intuitive interpretation is that e happens with likelihood p provided that it is enabled. Thus, p is a conditional probability. A partial mapping π is assumed that decorates an event with a probability in $(0, 1)$. Some events are grouped into clusters in order to model a stochastic experiment.

Definition 2.2.1 (*Probabilistic Event Structure, pes for Short*). A probabilistic event structure Π is a tuple $(E, \sharp, \mapsto, l, \pi)$ with

- E , a set of events,
- $\sharp \subseteq E \times E$, the irreflexive and symmetric conflict relation,
- $\mapsto \subseteq \mathcal{P}(E) \times E$, the bundle relation,
- $l : E \rightarrow \Omega$, the action labelling function,

- $\pi : E \longrightarrow_P (0, 1)$ the probability function,
 such that for any bundle-set X and event $e \in \text{dom}(\pi)$
- (1) $(X \times X) \setminus \text{Id}_E \subseteq \#$, and
 - (2) $\exists Q \subseteq \text{dom}(\pi) : (e \in Q) \wedge (Q \text{ is a cluster}) \wedge (\sum_{e' \in Q} \pi(e') = 1)$.

Here $\mathcal{P}(\cdot)$ denotes the power-set function, $\text{Id}_E = \{(e, e) \mid e \in E\}$, \longrightarrow_P indicates a partial function, $\text{dom}(\pi)$ denotes the domain of π , and a cluster is defined as follows:

$Q \subseteq E$ is called a *cluster*, if

- (i) $|Q| > 1$,
- (ii) $\forall e \in Q : l(e) = \tau$,
- (iii) $\forall e, e' \in Q : (e \neq e') \Rightarrow (e \# e')$,
- (iv) $\forall e \in Q, e' \in E : (e \# e') \Rightarrow (e' \in Q)$,
- (v) $\forall e, e' \in Q, X \subseteq E : (X \mapsto e) \Rightarrow (X \mapsto e')$.

$|Q|$ denotes the number of the elements of Q . Constraint (i) requires a cluster to consist of more than one event. This is convenient for technical reasons and poses no real practical constraint. In order to guarantee that stochastic experiments represented by clusters are indeed independent from their context, Constraint (ii) requires all events in a cluster to be internal, i.e. labelled with action τ . In this way, we are sure that such events are not subject to interaction anymore, which would make their probability dependent on the context in which they are embedded. According to Constraint (iii), events in a cluster mutually exclude each other such that only one event can happen. This is because the realization of a stochastic experiment usually has a single outcome. Constraint (iv) requires that events in a cluster are not in conflict with events outside the cluster. Allowing such conflicts will destroy the interpretation that an event probability represents the likelihood that this event happens once enabled. Finally, Constraint (v) requires all events in a cluster must be pointed to by the same bundle-sets. Together with the fourth constraint, this guarantee that if an event in a cluster is enabled all events in this cluster are enabled.

Constraint (1) in the definition of pes's enables us to uniquely define a causal ordering between the events in a system run. Constraint (2) requires the domain of π to be partitioned into clusters such that the sum of the probabilities assigned to all events in a cluster equals one. In this way, a cluster Q can be considered to represent a stochastic experiment for which the probability of outcome $e \in Q$ equals $\pi(e)$.

A pes is depicted as follows: Events are denoted by dots. Near the dot the action label is given. $e \# e'$ is indicated by a dotted line between e and e' . A bundle $X \mapsto e$ is indicated by an arrow to e and to this arrow each event in X is connected via a line. The probability of an event is depicted near to the event. A cluster is indicated by a shaded surface. When no confusion arises, we identify an event with its action label.

Example 2.2.2. Figure 1 is a pes. It can be understood as a system of message sender with probabilistic information, where $r = \text{read-message}$, $a = \text{use-}$

$channel-1$, $b = use-channel-2$, $c = use-channel-3$, $s = send-message$, and the internal actions indicate the system chooses the corresponding channels. The probability of choosing channel-1 is 0.28, the probability of choosing channel-2 is 0.42, and the probability of choosing channel-3 is 0.3.

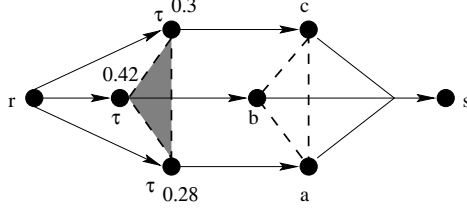


Figure 1. An example pes

By *PES* we denote the set of pes's. Elements of *PES* are denoted by Π and Π_k , where $\Pi = (E, \sharp, \mapsto, l, \pi)$ and $\Pi_k = (E_k, \sharp_k, \mapsto_k, l_k, \pi_k)$. When necessary, we also use E_Π , \sharp_Π , \mapsto_Π , l_Π and π_Π to represent the components of Π . Without loss of generality, we assume $E_f \cap E_g = \emptyset$ for two different pes's Π_f and Π_g .

We use $init(\Pi)$ to denote the set of initial events (of Π), and $exit(\Pi)$ denotes the set of successful termination events:

$$init(\Pi) = \{e \in E \mid \neg(\exists X \subseteq E : X \mapsto e)\}, exit(\Pi) = \{e \in E \mid l(e) = \surd\}.$$

To describe system runs of a pes Π , we first consider its underlying set of events. Suppose that γ is a finite sequence e_1, \dots, e_n of events of Π , where e_i and e_j are distinct whenever $i \neq j$. By $E(\gamma)$ we denote the event set $\{e_1, \dots, e_n\}$. Sequence $\gamma_{i-1} = e_1, \dots, e_{i-1}$ represents the $(i-1)$ -th prefix of γ .

In a system run any two events that occur should not be in conflict, and if a non-initial event happens in a system run then events that cause it should have happened before. Formally

$$en(\gamma_{i-1}) = \{e \in E \setminus E(\gamma_{i-1}) \mid (\forall e_j \in E(\gamma_{i-1}) : \neg(e \sharp e_j)) \wedge (\forall X \mapsto e : X \cap E(\gamma_{i-1}) \neq \emptyset)\}.$$

is the set of events that are enabled after the performance of γ_{i-1} .

Definition 2.2.3 (*Configuration*). If the event sequence $\gamma = e_1, \dots, e_n$ satisfies the condition $e_i \in en(\gamma_{i-1})$ for $1 \leq i \leq n$, then the set $E(\gamma)$ of all events occurring in γ is called a configuration of Π .

By $C(\Pi)$ we denote the set of all configurations of Π . Let σ be a configuration of Π . Event $e \in \sigma$ is said to be *maximal* in σ if for any bundle-set X and $e' \in E$, if $e \in X$ and $X \mapsto e'$ then $e' \notin \sigma$. We say that σ *successfully terminates* if there exists $e \in \sigma$ such that e is labelled with the successful termination action \surd . Π is called *well-labelled* if $\sigma \cap exit(\Pi)$ is empty or a singleton for any configuration σ of Π .

A system run may be represented as an equivalence class of labelled partial orders (pomsets). We define in the following a linear-time equivalence, termed pomset trace equivalence, on PES . A branching-time equivalence, termed history preserving bisimulation equivalence (see e.g. [10]), can be defined similarly, which further records where choices are made. Here we only describe the former equivalence relation in detail.

Assume that σ is a configuration of Π , and $e_i, e_j \in \sigma$. By $e_j \mapsto |_{\sigma} e_i$ we mean that there exists a bundle-set X such that $e_j \in X$ and $X \mapsto e_i$. Without confusion, we use $\rightarrow |_{\sigma}$ again to represent the transitive closure of $\mapsto |_{\sigma}$. We use $l|_{\sigma}$ as the restriction of l on σ , i.e. $l|_{\sigma}(e) = l(e)$ for $e \in \sigma$.

A configuration σ_1 of Π_1 and a configuration σ_2 of Π_2 are said to be *isomorphic*, denoted $\sigma_1 \approx \sigma_2$, if there exists a bijection $h : \sigma_1 \rightarrow \sigma_2$, such that for arbitrary $e, e' \in \sigma_1$,

- (1) $e \rightarrow_1 |_{\sigma_1} e'$ iff $h(e) \rightarrow_2 |_{\sigma_2} h(e')$,
- (2) $l_1|_{\sigma_1}(e) = l_2|_{\sigma_2}(h(e))$,
- (3) $e \in \text{dom}(\pi_1)$ and $\pi_1(e) = p$ iff $h(e) \in \text{dom}(\pi_2)$ and $\pi_2(h(e)) = p$.

When $\sigma_1 \approx \sigma_2$, σ_1 and σ_2 only differ in the event names, i.e. the action labels, causality relations as well as the probabilities of events remain the same.

By $C(\Pi_1) \approx C(\Pi_2)$ we denote the fact that for any configuration of Π_1 , there is an isomorphic configuration of Π_2 , and vice versa.

Definition 2.2.4 (*Pomset Trace Equivalence*). Π_1 and Π_2 are pomset trace equivalent, denoted $\Pi_1 \approx_p \Pi_2$, if $C(\Pi_1) \approx C(\Pi_2)$.

A Denotational Semantics. We first describe the operators on PES corresponding to those in the probabilistic process algebra. Here we only present the definition of probabilistic choice $+_p$ in detail. For the definitions of *action prefix* $a_T.$, *abstraction* $\backslash A$, *relabelling* $[\lambda]$, *sequential composition* $;$, *parallel composition* \parallel_A and *choice* $+$, we refer the reader to [5,6,15,16,17].

Probabilistic Choice. $\Pi_1 +_p \Pi_2 = (E_1 \cup E_2, \sharp, \mapsto_1 \cup \mapsto_2, l_1 \cup l_2, \pi)$, where
 $\sharp = \sharp_1 \cup \sharp_2 \cup (\text{init}(\Pi_1) \times \text{init}(\Pi_2)) \cup (\text{init}(\Pi_2) \times \text{init}(\Pi_1))$,
 $\pi = \pi_1|_{(E_1 \setminus \text{init}(\Pi_1))} \cup \pi_2|_{(E_2 \setminus \text{init}(\Pi_2))} \cup \{(e, p) \mid e \in \text{init}(\Pi_1) \setminus \text{dom}(\pi_1)\}$
 $\cup \{(e, p \cdot \pi_1(e)) \mid e \in \text{init}(\Pi_1) \cap \text{dom}(\pi_1)\} \cup \{(e, 1 - p) \mid e \in \text{init}(\Pi_2) \setminus \text{dom}(\pi_2)\}$
 $\cup \{(e, (1 - p) \cdot \pi_2(e)) \mid e \in \text{init}(\Pi_2) \cap \text{dom}(\pi_2)\}$.

In this definition, we assume events in $\text{init}(\Pi_1)$ and $\text{init}(\Pi_2)$ are all labelled with internal τ -actions. This assumption guarantees that $\Pi_1 +_p \Pi_2$ remains a pes, and has no influence on providing semantics for the probabilistic process algebra. $\pi_k|_{(E_k \setminus \text{init}(\Pi_k))}$ represents the restriction of π_k on $E_k \setminus \text{init}(\Pi_k)$ ($k = 1, 2$).

An expression is called *closed*, if every occurrence of a process variable x is in the range of a μx -operator. A recursion $\mu x.B_1$ is said to be *guarded*, if

B_1 becomes guarded by substituting for a finite number of times the bodies of processes for the process instantiations occurring in B_1 [15].

We want to use the cpo-based denotational semantics as proposed in [15]. For this we have to require that all the recursions occurring in the given closed expression B are guarded [15]. Let s denote this semantics. The semantic model $s(B)$ of expression B is a pes, which is derived as follows:

$$\begin{aligned} s(0) &= (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset), \\ s(1) &= (\{e\}, \emptyset, \emptyset, \{(e, \sqrt{\})\}, \emptyset), \\ s(\circ B_1) &= \circ s(B_1) \text{ for } \circ \in \{a_T., \setminus A, [\lambda]\}, \\ s(B_1 \circ B_2) &= s(B_1) \circ s(B_2) \text{ for } \circ \in \{;, \parallel_{\sqrt{}}, +, +_p\}, \end{aligned}$$

and for guarded recursion $(\mu x.B_1)$, $s(\mu x.B_1)$ is defined as the least upper bound of a set of pes's with a complete partial order (cpo) [1,8,15].

When $s(B)$ is mentioned in the following, we always suppose that expression B is closed and all the recursions occurring in B are guarded.

Example 2.2.5. The semantic model $s(B)$ of the expression B of Example 2.1.3 is pomset trace equivalent to the pes Π of Example 2.2.2 (Figure 1). That is, $s(B) \cong_p \Pi$.

3 Action Refinement

In this section, we discuss syntactic action refinement operation in the probabilistic process algebra. The semantic counterpart is also investigated.

3.1 Syntactic Refinement

Firstly, we have to sort out some 'bound' actions. For a given expression B , let

$$\text{Sort}(B) = \cup\{A \mid \setminus A \text{ occurs in } B \text{ or } A = \text{dom}(\lambda) \text{ and } [\lambda] \text{ occurs in } B\}.$$

All actions in $\text{Sort}(B)$ are not allowed to be refined, since they may lead to a confusion of the communication levels like in the unprobabilistic case [13].

Assume that Ω_0 is a subset of Ω , representing the set of actions that need not or cannot be refined such that $\text{Sort}(B) \cup \{\tau, \sqrt{\}\} \subseteq \Omega_0$. Let $g : \Omega \setminus \Omega_0 \rightarrow PPA$ be a function. For convenience, in the following we also use $A_{g(a)}$ to denote the action singleton $\{a\}$ when $a \in \Theta \cap \Omega_0$.

Definition 3.1.1 (*Refinement Function for Expression B*).

g is a refinement function for $0, 1$ and x ,

g is a refinement function for $a.B_1$ iff g is a refinement function for B_1 ,

g is a refinement function for $B_1 \circ B_2$ iff g is a refinement function for B_1 and B_2 , where $\circ \in \{;, +, +_p\}$,

g is a refinement function for $B_1 \parallel_A B_2$ iff g is a refinement function for B_1 and B_2 , and for any two distinct $a \in A$ and $b \in \Theta$, $A_{g(a)} \cap A_{g(b)} = \emptyset$,

g is a refinement function for $\circ B_1$ iff g is a refinement function for B_1 , and for any $a \in \Omega \setminus \Omega_0$, $A_{g(a)} \cap \text{Sort}(P) = \emptyset$, where $\circ \in \{\setminus A, [\lambda]\}$,
 g is a refinement function for $\mu x.B_1$ iff g is a refinement function for B_1 .

We forbid the actions in $\text{Sort}(B)$ to occur in $g(a)$, and we pose one more constraint in the definition for parallel composition. Usually, like the unprobabilistic case such constraints cannot be avoided for refinement on syntactic level. Otherwise they may result in a confusion of the communication levels [13].

Example 3.1.2. Suppose that B and B_c are the expressions of Example 2.1.3, and $\Omega_0 = \Omega \setminus \{c\}$. Let $g : c \mapsto B_c$. Then g is a refinement function for B .

Let g be a refinement function for expression B . Now the question is how g can be applied to B to obtain a refined expression. Our basic idea is that the refined expression of $a.B_1$ where $a \notin \Omega_0$ is defined as the sequential composition of $\tau.g(a)$ and the refined expression of B_1 . This construction guarantees an equivalence between syntactic and semantic refinement. An intuitive explanation of the prefix τ can be given as follows: We may assume that every process has a start point that executes an internal silent action, the prefix τ here is used to model such a start point of the process $g(a)$ used to substitute action a .

Definition 3.1.3 (*Syntactic Refinement of Expression B*). The refinement $g(B)$ of expression B is defined as follows:

$$\begin{aligned} g(0) &= 0, g(1) = 1, g(x) = x, \\ g(a.B_1) &= \begin{cases} a.g(B_1) & \text{if } a \in \Omega_0, \\ (\tau.g(a)); g(B_1) & \text{otherwise,} \end{cases} \\ g(\circ B_1) &= \circ g(B_1) \text{ for } \circ \in \{\setminus A, [\lambda]\}, \\ g(B_1 \circ B_2) &= g(B_1) \circ g(B_2) \text{ for } \circ \in \{;, +, +_p\}, \\ g(B_1 \parallel_A B_2) &= g(B_1) \parallel_{g(A)} g(B_2) \text{ where } g(A) = \cup_{a \in A} A_{g(a)}, \\ g(\mu x.B_1) &= \mu x.g(B_1). \end{aligned}$$

Theorem 3.1.4. Suppose that $B \in PPA$ and g is a refinement function for B . Then $g(B) \in PPA$.

Furthermore, $g(B)$ is closed if B is closed. All the recursions occurring in $g(B)$ are guarded if all the recursions occurring in B are guarded.

Example 3.1.5. Consider B and g of Example 3.1.2.

$$g(B) = r.((\tau.a.s.0 +_{0.4} \tau.b.s.0) +_{0.7} \tau.((\tau.c_1.1 +_{0.8} \tau.c_2.1)); (s.0)).$$

3.2 Semantic Refinement

The refinement methodology developed for various event structures in e.g. [10] does not work for pes's. For example, when we refine action a in the pes

$a \bullet \dots \bullet \tau$ by the pes of Figure 2(a), the resulting Figure 2(b) is no longer a pes because the property of clusters is violated. Our solution is to introduce a special event in each pes used to substitute an action.

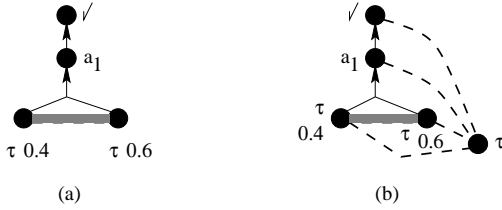


Figure 2. An example showing a refinement methodology does not work

We call $\tau.H$, denoted $r(H)$, the *rooted pes* associated with H . In $r(H)$, the newly introduced event that corresponds to the prefix τ is called the *start-event* of H , and is denoted by $o_{r(H)}$. This new event resembles the special events used in [4,9,22,24,25], and can be understood as the start point of system, which is usually supposed to be executing an internal silent action. In the definition of syntactic refinement, we introduced the prefix τ for $g(a)$ with the intention to obtain an 'equivalence result' for syntactic and semantic refinement.

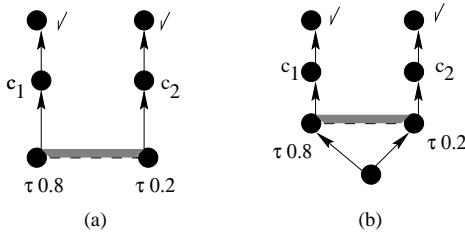


Figure 3. A pes and the rooted pes associated with it

Example 3.2.1. Let H_c be the pes of Figure 3(a). Then $r(H_c)$ is the pes of Figure 3(b). The pes H_c of Figure 3(a) is in fact the semantic model of expression B_c of Example 2.1.3. Namely $s(B_c) = H_c$. In the following, we want to replace action c in the pes of Example 2.2.2 (Figure 1) by H_c . We may understand that in this example c_1 and c_2 are two new channels that can be used to decompose c . Similarly, the internal actions indicate the system chooses the corresponding channels. The probabilities of choosing channel c_1 and channel c_2 are 0.8 and 0.2, respectively.

The refinement of an action a , say H_a , should be a pes. Since action \sqrt only represents successful termination of a system, a system run, if it is not a deadlock, should contain exactly one \sqrt -event. So we require that H_a is well-labelled.

Definition 3.2.2 (*Semantic Refinement Function*). $f : \Omega \setminus \Omega_0 \rightarrow PES$ is called a *refinement function*, if $f(a)$ is well-labelled for any action $a \in \Omega \setminus \Omega_0$.

We say that $f(a)$ is a *refinement of action a*. In practice, it is reasonable to require that $f(a)$ is finite. If so the condition in this definition can be checked and easily enforced.

Example 3.2.3. Suppose $\Omega \setminus \Omega_0 = \{c\}$, and Π_c is the pes of Example 3.2.1. Then $f(c) = \Pi_c$ is a refinement of action c .

Let f be a semantic refinement function. Now we see how f can be used to refine a pes. For simplicity, we use $rfl(e)$ and $rf(a)$ to abbreviate $r(f(l(e)))$ and $r(f(a))$, respectively.

Definition 3.2.4 (*Refinement of a pes*). The *refinement of Π* is defined as $f(\Pi) = (E_f, \#_f, \mapsto_f, l_f, \pi_f)$, where

- $E_f = \{(e, e') \mid (e \in E) \wedge (l(e) \notin \Omega_0) \wedge (e' \in E_{rfl(e)})\} \cup \{(e, e) \mid (e \in E) \wedge (l(e) \in \Omega_0)\}$,
- $\forall (e_1, e_2), (e'_1, e'_2) \in E_f, (e_1, e_2) \#_f (e'_1, e'_2)$ iff
 - if $(e_1 = e'_1)$ then $(e_2 \#_{rfl(e_1)} e'_2) \vee (e_2, e'_2 \in \text{exit}(rfl(e_1)) \wedge e_2 \neq e'_2)$,
 - if $(e_1 \neq e'_1)$ then
 - if $(e_2 = e_1) \wedge (e'_2 = e'_1)$ then $(e_1 \# e'_1)$,
 - if $(e_2 \neq e_1) \wedge (e'_2 = e'_1)$ then $(e_1 \# e'_1) \wedge (e_2 \in \{o_{rfl(e_1)}\} \cup \text{exit}(rfl(e_1)))$,
 - if $(e_2 = e_1) \wedge (e'_2 \neq e'_1)$ then $(e_1 \# e'_1) \wedge (e'_2 \in \{o_{rfl(e'_1)}\} \cup \text{exit}(rfl(e'_1)))$,
 - if $(e_2 \neq e_1) \wedge (e'_2 \neq e'_1)$ then $(e_1 \# e'_1) \wedge ((e_2 = o_{rfl(e_1)} \wedge e'_2 = o_{rfl(e'_1)}) \vee (e_2 \in \text{exit}(rfl(e_1)) \wedge e'_2 \in \text{exit}(rfl(e'_1))))$,
- $\forall X \subseteq E_f, (e_1, e_2) \in E_f, X \mapsto_f (e_1, e_2)$ iff
 - if $(e_2 \neq e_1) \wedge (e_2 \in E_{rfl(e_1)} \setminus \{o_{rfl(e_1)}\})$ then

$$(\rho_1(X) = \{e_1\}) \wedge (\rho_2(X) \mapsto_{rfl(e_1)} e_2),$$
 - if $(e_2 \neq e_1 \wedge e_2 = o_{rfl(e_1)}) \vee (e_2 = e_1)$ then $(\rho_1(X) \mapsto e_1)$

$$\wedge (\rho_2(X) = \cup_{e \in \rho_1(X), l(e) \notin \Omega_0} \text{exit}(rfl(e)) \cup (\cup_{e \in \rho_1(X), l(e) \in \Omega_0} \{e\})),$$
- $\forall (e_1, e_2) \in E_f,$
 - if $(e_2 \neq e_1)$ then
 - if $(e_2 \notin \text{exit}(rfl(e_1)))$ then $(l_f(e_1, e_2) = l_{rfl(e_1)}(e_2))$,
 - if $(e_2 \in \text{exit}(rfl(e_1)))$ then $(l_f(e_1, e_2) = \tau)$,
 - if $(e_2 = e_1)$ then $(l_f(e_1, e_2) = l(e_1))$,
- $\forall (e_1, e_2) \in E_f, (e_1, e_2) \in \text{dom}(\pi_f)$ iff
 - $(e_1 \in \text{dom}(\pi)) \vee ((e_1 \neq e_2) \wedge (e_2 \in \text{dom}(\pi_{rfl(e_1)})))$,
 - if $(e_1 \in \text{dom}(\pi))$ then $(\pi_f(e_1, e_2) = \pi(e_1))$,
 - if $(e_1 \neq e_2) \wedge (e_2 \in \text{dom}(\pi_{rfl(e_1)}))$ then $(\pi_f(e_1, e_2) = \pi_{rfl(e_1)}(e_2))$.

Here $\rho_1(X) = \{e \mid (e, e') \in X\}$ and $\rho_2(X) = \{e' \mid (e, e') \in X\}$.

Theorem 3.2.5. Suppose that $\Pi \in PES$ and f is a refinement function. Then $f(\Pi) \in PES$.

Example 3.2.6. Let Π be the pes of Example 2.2.2 (Figure 1) and $f(c)$ be the refinement of action c given in Example 3.2.3 (Figure 3(a)). Then $f(\Pi)$ is the pes of Figure 4.

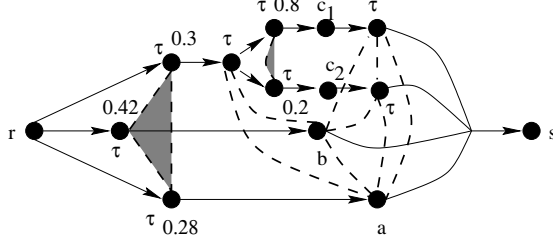


Figure 4. The refined pes

4 Properties

Hereby, suppose that $g : \Omega \setminus \Omega_0 \rightarrow PPA$ is a syntactic refinement function for a given expression B , and $f : a \mapsto s(g(a))$ ($a \in \Omega \setminus \Omega_0$) is a semantic refinement function. Π, Π_1 and Π_2 are pes's, and f_1 and f_2 are two semantic refinement functions.

Let σ be a configuration of Π , $e \in \sigma$ with $l(e) \notin \Omega_0$, and σ_e a configuration of pes $rfl(e)$ which is used to substitute for that action labelled to event e . Furthermore, assume $rfl(e)$ satisfies the condition that σ_e successfully terminates if event e is not maximal in σ . Let

$$\sigma_g = \{(e, e_j) \mid e \in \sigma \text{ and if } l(e) \in \Omega_0 \text{ then } e_j = e \text{ otherwise } e_j \in \sigma_e\}.$$

We call σ_g a *refinement of configuration* σ . It is derived by replacing each event e in configuration σ with $l(e) \notin \Omega_0$ by a configuration σ_e of $rfl(e)$.

Theorem 4.1.1 (Safety). $C(f(\Pi)) = \{\sigma_f \mid \sigma_f \text{ is a refinement of } \sigma \in C(\Pi)\}$.

It is natural to use $rfl(e)$ rather than $f(l(e))$ to substitute action $l(e)$ in Π . The start-event we introduced is to imitate the start point of system. Theorem 4.1.1 demonstrates that the refinement of a configuration of Π can be obtained by replacing each event e with $l(e) \notin \Omega_0$ in this configuration by a configuration of $rfl(e)$.

It is straightforward to see that the causality relations in each σ_e remain unchanged in the refinement of σ . The causality relations in σ remain unchanged

in the meaning that if e causes e' in σ , then the maximal event in σ_e causes the minimal event in $\sigma_{e'}$. Furthermore, the probability of $(e, e_j) \in \sigma_g$ equals the probability of $e \in \sigma$ when $e \in \text{dom}(\pi)$ and equals the probability of $e_j \in \sigma_e$ when $e_j \in \text{dom}(\pi_{f(l(e))})$.

The behaviour of a refined system can be thus inferred compositionally from the behaviour of the original system and from the behaviour of those substituted for the actions. Hence, our refinement is safe.

Theorem 4.1.2 (Congruence Result). If $\Pi_1 \cong_p \Pi_2$ and $f_1(a) \cong_p f_2(a)$ for any $a \in \Omega \setminus \Omega_0$, then $f_1(\Pi_1) \cong_p f_2(\Pi_2)$.

This theorem indicates that pomset trace equivalence is a congruence under our refinement.

The start-events can be neglected by considering only the observable behaviour of systems. On the other hand, when dealing with the coincidence problem, we have to consider such behaviour. Namely we have to abstract from the internal actions.

Let σ be a configuration of Π . Then $\{e \in \sigma \mid l(e) \neq \tau\}$ is called an *observational configuration* of Π . This observational configuration is said to be derived from configuration σ .

Assume that σ_0 is a configuration of Π , and $e_i, e_j \in \sigma_0$. For the observational configuration σ derived from σ_0 , a binary relation $\rightarrow|_\sigma$ on σ is defined as the relation $\mapsto|_{\sigma_0}$ in which the causally necessary internal τ -events that occurs are neglected. That is, $e_j \rightarrow|_\sigma e_i$ iff $e_j \mapsto|_{\sigma_0} e_i$ or there exist $e_{k_p} \in \sigma$ with $l(e_{k_p}) = \tau$ for $p = 1, \dots, m$, such that $e_j \mapsto|_{\sigma_0} e_{k_1}$, $e_{k_q} \mapsto|_{\sigma_0} e_{k_{q+1}}$ ($q = 1, \dots, m - 1$) and $e_{k_m} \mapsto|_{\sigma_0} e_i$. It is actually the causality relation that is observable in σ_0 . In the sequel, we use $\rightarrow|_\sigma$ again to represent the transitive closure of $\rightarrow|_\sigma$.

As in Section 2.2, we can similarly define that an observational configuration of Π_1 and an observational configuration of Π_2 are *isomorphic*, and we say that Π_1 and Π_2 are *observational pomset trace equivalent*, denoted $\Pi_1 \cong_{op} \Pi_2$, if for any observational configuration of Π_1 , Π_2 has an observational configuration isomorphic to it, and vice versa. Clearly, $\Pi_1 \cong_p \Pi_2$ implies $\Pi_1 \cong_{op} \Pi_2$.

Theorem 4.1.3 (Coincidence Result). $f(s(B)) \cong_{op} s(g(B))$.

This theorem demonstrates that with respect to the cpo-based denotational semantics our syntactic and semantic refinement operations coincide up to observational pomset trace equivalence.

Please note that all the recursions occurring in expression B should be guarded in Theorem 4.1.3. In addition, Theorem 4.1.3 may not hold if we do not abstract away from τ -events. The main reason is that we can by no means require that some τ -actions can be executed simultaneously. The constraint that actions in $\text{Sort}(B)$ are not allowed to be refined and the constraint for refinement of parallel composition are also necessary.

Example 4.1.4. The semantic model of the expression $g(B)$ of Example 3.1.5 is observational pomset trace equivalent to the pes $f(\Pi)$ of Example 3.2.5 (Figure 4). Therefore, up to observational pomset trace equivalence the refinement of pes Π can be used as the semantic model of the refinement of expression B , and on the other hand the refinement of expression B can be used as the specification of the refinement of pes Π .

If we require that the pes Π_a used to replace any observable action a remains 'observable', namely any configuration of Π_a that successfully terminates contains at least one event which is labelled with an observable action, then Theorem 4.1.1 holds for observational configurations and Theorem 4.1.2 holds for observational pomset trace equivalence.

All the conclusions above hold for history preserving bisimulation equivalence and observational history preserving bisimulation equivalence.

5 Concluding Remarks

In this paper, we defined the notions of syntactic and semantic action refinements in a probabilistic process algebra with a true concurrency model. We showed that they coincide with each other up to observational pomset trace equivalence and observational history preserving bisimulation equivalence with respect to a cpo-based denotational semantics. We demonstrated that they are safe and (observational) pomset trace equivalence and history preserving bisimulation equivalence are congruences under them.

This paper is a further development of [10]. The refinement operation defined here can be mapped in a safe refinement operation in the non-deterministic counterpart. Moreover, The results presented in this paper also hold for the metric semantics developed in [19]. This semantics can be easily extended to the probabilistic case.

Appendix

In this part we sketch proofs of our theorems.

Theorems 3.1.4 follows from the fact that $ppa(B) = true$ implies $ppa(g(B)) = true$. Theorem 3.2.5 can be proved by directly checking the two conditions in the definition of pes's. We hereby focus on the remaining statements.

Lemma 1 below shows that introducing the start-event into a pes does not have influence on the observational behaviours. It does not alter the properties of being well-labelled.

Lemma 1. (1) Π is well-labelled iff $r(\Pi)$ is well-labelled,
 (2) $\Pi_1 \cong_p \Pi_2$ iff $r(\Pi_1) \cong_p r(\Pi_2)$.

Now suppose that σ_f is a configuration of $f(\Pi)$. Let

$$\rho_1(\sigma_f) = \{e \mid \exists(e, e_j) \in \sigma_f\}.$$

For $e \in \rho_1(\sigma_f)$ with $l(e) \notin \Omega_0$, let

$$\rho_2(\sigma_f, e) = \{e_j \mid \exists(e, e_j) \in \sigma_f\}.$$

$\rho_1(\sigma_f)$ is the projection of σ_f on Π , and $\rho_2(\sigma_f, e)$ the projection of σ_f on $rfl(e)$.

Lemma 2. (1) $\rho_1(\sigma_f) \in C(\Pi)$, (2) $\rho_2(\sigma_f, e) \in C(rfl(e))$ and it successfully terminates if e is not maximal in $\rho_1(\sigma_f)$.

The semantic models of expressions have the following properties:

- σ_1 is a configuration of $s(B_1)$ iff $\sigma = \{e_a\} \cup \sigma_1$ is a configuration of $s(a_T.B_1)$, where event e_a is labelled with action a . The causality relations, action labels and probabilities of events in σ_1 remains unchanged in σ .

- σ is a configuration of $s(B_1; B_2)$ iff either σ is a configuration of $s(B_1)$, or $\sigma = \sigma_1 \cup \sigma_2$, where σ_1 is a configuration of $s(B_1)$ that successfully terminates and σ_2 is a configuration of $s(B_2)$. The causality relations, action labels and probabilities of events in σ_1 and σ_2 remain unchanged in σ except that the event in σ_1 labelled with action \surd is relabelled with action τ in σ .

- σ is a configuration of $s(B_1 + B_2)$ iff it is a configuration of $s(B_1)$ or a configuration of $s(B_2)$. The causality relations, action labels and probabilities of events remains unchanged.

- σ is a configuration of $s(B_1 +_p B_2)$ iff it is a configuration of $s(B_1)$ or a configuration of $s(B_2)$. The causality relations and action labels remains unchanged. If e is an initial event and its probability is q in $s(B_1)$ [$s(B_2)$], then the probability of e in $s(B_1 +_p B_2)$ is $q \cdot p$ [resp. $q \cdot (1 - p)$].

- For a configuration σ of $s(B_1 \parallel_A B_2)$, let

$$\begin{aligned} \rho_1(\sigma) &= \{e_1 \mid \exists(e_1, e_2) \in \sigma \wedge e_1 \neq *\} \text{ and} \\ \rho_2(\sigma) &= \{e_2 \mid \exists(e_1, e_2) \in \sigma \wedge e_2 \neq *\} \end{aligned}$$

then $\rho_k(\sigma)$ is a configuration of $s(B_k)$ ($k \in \{1, 2\}$). On the other hand, assume that σ_k is a configuration of $s(B_k)$ which satisfies the conditions:

- (1) $e_{k1}^a, \dots, e_{kn}^a$ are all the events in σ_k labelled with action $a \in A$, and
- (2) if $e_{1i}^a \mapsto_1 e_{1j}^a$ then $e_{2j}^a \not\mapsto_2 e_{2i}^a$ and if $e_{2i}^a \mapsto_2 e_{2j}^a$ then $e_{1j}^a \not\mapsto_1 e_{1i}^a$.

Then

$$\begin{aligned} &\{(e_1, *) \mid e_1 \in E(\sigma_1) \wedge \text{the action labelled to } e_1 \text{ is not in } A\} \cup \\ &\{(*, e_2) \mid e_2 \in E(\sigma_2) \wedge \text{the action labelled to } e_2 \text{ is not in } A\} \cup \\ &\cup_{a \in A} \{(e_{1i}^a, e_{2i}^a) \mid i = 1, \dots, n\} \end{aligned}$$

is a configuration of $s(B_1 \parallel_A B_2)$. In addition, event (e_1, e_2) causes event (e'_1, e'_2) in $s(B_1 \parallel_A B_2)$ iff event e_1 causes event e'_1 in $s(B_1)$ or event e_2 causes event e'_2 in $s(B_2)$ (See e.g. [15,17] for the usage of *-event).

- σ_1 is a configuration of $s(B_1)$ iff it is a configuration of $s(B_1[\lambda])$, but each event $e \in E(\sigma_1)$ is relabelled by λ . The causality relations remain unchanged.
- σ_1 is a configuration of $s(B_1)$ iff it is a configuration of $s(B_1 \setminus A)$, but each event $e \in E(\sigma_1) \cap A$ is relabelled with τ . The causality relations remain unchanged.
- σ is a configuration of $s(\mu x.B_1)$ iff there exists $k \geq 0$ such that σ is a configuration of $v_{B_1}^k(\perp)$. The causality relations remain unchanged. Here $\text{pes } \perp = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$, function $v_{B_1} : PES \rightarrow PES$ is defined that substitutes a pes for each occurrence of x in B_1 , and $v_{B_1}^k(\perp) = v_{B_1}(v_{B_1}^{k-1}(\perp))$.

An immediate corollary of these facts is the following lemma.

Lemma 3. If $s(B_1) \cong_{op} s(B_2)$ and $s(B_3) \cong_{op} s(B_4)$ then
 $\circ s(B_1) \cong_{op} \circ s(B_2)$ for $\circ \in \{a_T., \setminus A, [\lambda]\}$,
 $s(B_1 \circ B_3) \cong_{op} s(B_2 \circ B_4)$ for $\circ \in \{;, \|_A, +, +_p\}$.

Lemma 4 follows from the above facts, Lemma 3 and the requirement that the actions involved with abstraction and relabelling operators are not allowed to be refined.

Lemma 4. $C(f(a.s(B_1))) = C(a.f(s(B_1)))$ if $a \in \Omega_0$,
 $C(f(a.s(B_1))) = C((\tau.f(a)); f(s(B_1)))$ if $a \notin \Omega_0$,
 $C(f(\circ s(B_1))) = C(\circ f(s(B_1)))$ for $\circ \in \{\setminus A, [\lambda]\}$,
 $C(f(s(B_1) \circ s(B_2))) = C(f(s(B_1)) \circ f(s(B_2)))$ for $\circ \in \{;, +, +_p\}$.

By the facts above, Lemma 4 and the condition that $A_{g(a)} \cap A_{g(b)} = \emptyset$ for two distinct $a \in A$ and $b \in \Theta$, for parallel composition we have the following lemma. Remark that it does not hold if τ -events are not abstracted away.

Lemma 5. $f(s(B_1) \|_A s(B_2)) \cong_{op} f(s(B_1)) \|_{g(A)} f(s(B_2))$.

Theorem 4.1.1 follows from Lemma 1 and Lemma 2. Theorem 4.1.2 follows from Lemma 2 and Theorem 4.1.1. Theorem 4.1.3 follows from Lemmas 3, 4 and 5. The conclusion for recursion follows from the fact that $f(v_{B_1}^k(\perp)) \cong_{op} v_{g(B_1)}^k(\perp)$ for $k \geq 0$. This fact can be proved via the conclusions for the other operators.

Acknowledgements

Thanks to the anonymous referees for their valuable comments and suggestions.

References

1. Abramsky, S., Jung, A., Domain Theory, *Handbook of Logic in Computer Science* (S. Abramsky and Dov M. Gabbay eds.), Clarendon Press, Volume 3: 1 – 168, 1994.
2. Aceto, L, *Action Refinement in Process Algebra*, Cambridge Univ. Press, 1992.

3. Bolognesi, T., Brinksma, E., Introduction to the ISO Specification Language LOTOS, *Comp. Netw. and ISDN Syst.*, 14: 25 – 59, 1987.
4. Bowman, H., Derrick, J., Extending LOTOS with Time: A True Concurrency Perspective, *Lecture Notes in Computer Science*, 1231: 383 – 399, 1997.
5. Bowman, H., Katoen, J.-P., A True Concurrency Semantics for ET-LOTOS, *Proceedings Int. Conference on Applications of Concurrency to System Design*, 228 – 239, 1998.
6. Brinksma, E., Katoen, J.-P., Langerak, R., and Latella, D., Partial-Order Models for Quantitative Extensions of LOTOS, *Computer Networks and ISDN Systems*, 30(9/10): 925 – 950, 1998.
7. D'Argenio, P. R., Hermanns, H., and Katoen, J.-P., On Generative Parallel Composition, *Electronic Notes in Theoretical Computer Science*, 22, 1999.
8. Fecher, H., Majster-Cederbaum, M., and Wu, J., Bundle Event Structures: A Revised Cpo Approach, *Information Processing Letters*, accepted, 2001.
9. Fecher, H., Majster-Cederbaum, M., and Wu, J., Refinement of Actions in a Real-Time Process Algebra with a True Concurrency Model, *REFINE 2002*, accepted, 2002.
10. van Glabbeek, R., Goltz, U., Refinement of Actions and Equivalence Notions for Concurrent Systems, *Acta Informatica*, 37: 229 – 327, 2001.
11. van Glabbeek, R., Smolka, S. A., and Steffen, B., Reactive, Generative and Stratified Models of Probabilistic Processes, *Information and Computation*, 121: 59 – 80, 1995.
12. van Glabbeek, R., Smolka, S. A., Steffen, B., and Tofts, C. M. N., Reactive, Generative and Stratified Models of Probabilistic Processes, *Proc. LICS'90*, IEEE-CS Press, 130 – 141, 1990.
13. Gorrieri, R., Rensink, A., Action Refinement, *Handbook of Process Algebra*, Elsevier Science, 1047 – 1147, 2001.
14. Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall, 1985.
15. Katoen, J.-P., *Quantitative and Qualitative Extensions of Event Structures*, PhD Thesis, University of Twente, 1996.
16. Katoen, J.-P., Baier, C., and Latella, D., Metric Semantics for True Concurrent Real Time, *Th. Comp. Sci.*, 254(1-2): 501 – 542, 2001.
17. Katoen, J.-P., Langerak, R., Latella, D., and Brinksma, E., On Specifying Real-Time Systems in a Causality-Based Setting, in: *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Lecture Notes in Computer Science, 1135: 385 – 405, 1996.
18. Langerak, R., *Transformations and Semantics for LOTOS*, PhD Thesis, University of Twente, 1992.
19. Loogn, R., Goltz, U., Modelling Nondeterministic Concurrent Processes with Event Structures, *Fund. Inf.*, 14(1): 39 – 74, 1991.
20. Majster-Cederbaum, M., Salger, F., Correctness by Construction: Towards Verification in Hierarchical System Development, *Lecture Notes in Computer Science*, 1885: 163 – 180, 2000.
21. Majster-Cederbaum, M., Salger, F., and Sorea, M., A Priori Verification of Reactive Systems, *Proc. FORTE/PSTV 2000*, Kluwer Academic Publishers, 35 – 50, 2000.
22. Majster-Cederbaum, M., Wu, J., Action Refinement for True Concurrent Real Time, *Proc. ICECCS 2001*, IEEE Computer Society Press, 58 – 68, 2001.
23. Milner, R., *Communication and Concurrency*, Prentice-Hall, 1989.
24. Murphy, D., Pitt, D., Real-Timed Concurrent Refineable Behaviours, *Lecture Notes in Computer Science*, 571: 529 – 545, 1992.
25. Žic, J., Time-Constrained Buffer Specifications in CSP+T and Timed CSP, *ACM Transactions on Programming and Systems*, 16(6): 1661 – 1674, 1994.