プログラミング-言語・基礎・実践- 16－11
(1994. 3. 10)

仕様合成オペレータを持つ並行処理記述言語のアクションリファインメント

小村 誠一　　平川 豊　　市川 晴久
{ komura, hirakawa, ichikawa }@sdesun.ntt.jp
NTT ソフトウェア研究所
〒180, 東京都 武蔵野市 緑町 3-9-11

あらまし　ソフトウェア仕様の発展には初期設計時の詳細化と保守・運用時の機能変更がある。詳細化手法は設計法における重要なコンセプトの一つであり、従来より多くの研究が行われている。一方、ユーザ層の拡大により、システムへの要求を初期設計時に把握することがますます困難となり、運用開始後の仕様変更機能の重要性が増している。このような仕様の発展をサポートするための基本的検討として、二つの仕様を合成するオペレータをもつ並行処理記述言語のアクションリファインメントについて考察する。双模倣同値な二つの仕様は一般にはアクションリファインメント後に双模倣同値にならない。本稿では、上記言語の双模倣同値な仕様がアクションリファインメント後、双模倣同値になるための十分条件を示す。

和文キーワード　プロセス代数、仕様合成、アクションリファインメント、双模倣同値。

Action refinement in a process algebraic language that has a specification synthesis operator

Seiichi KOMURA　Yutaka HIRAKAWA　Haruhisa ICHIKAWA
{ komura, hirakawa, ichikawa }@sdesun.ntt.jp
NTT Software Laboratories

3-9-11 Midori-Cho, Musashino-Shi, Tokyo 180, Japan

Abstract　System development process is repetition of perceiving new requirements and implementing them. When new requirements are clarified in the next design stage or after completing the design, current systems are considered to be incomplete. Incremental design method supports these repetition in design process. New specification can be added to an incomplete current specification, and a revised specification is easily synthesized. This article discusses theoretical foundations of incremental design methodologies in a process algebraic language. On the other hand, refinement is one of the most important concept in design method. However, it is known that refined specifications obtained from two bisimulation equivalent specifications are not guaranteed to be bisimulation equivalent. Unfortunately, it is same in incremental design methods. This article clarifies a pair of sufficient conditions to preserve bisimulation equivalence for incremental design method.

英文 key words　process algebra, specification synthesis, action refinement, bisimulation equivalence.

1 Introduction

In order to comply with increasing complexity of computer systems, efficient development methods are required. Many design methodologies, especially formal description techniques have therefore been proposed. Among the concepts of design methodologies, refinement is one of the most important. It supports so-called top-down design. Incremental development methodology that we have already proposed is a design concept which differs in standpoint from top-down design [IIKTS91]. In incremental design method, specification is always considered to be incomplete. New specification is added to incomplete current specification, and a revised specification is easily synthesized when new requirements are clarified. The advantage of incremental methodology is that it is easy to add new functions which are required after the systems have been put into use. Because of the increasing numbers and kinds of computer users, it is becoming difficult to recognize the requirements for computer systems. Therefore we believe that incremental enhancement of systems is becoming more important.

This article discusses the theoretical foundations for action refinement and specification synthesis in a process algebraic language. In the field of process algebra, there have been reports about action refinement [Itoh90] [Acet92] [Glab90]. However these did not consider specification synthesis. On the other hand, a process algebraic language that features a synthesis operator is proposed in [IYK90]. But that article did not discuss action refinement. So far, there is no report that investigates refinement and incremental design simultaneously.

It is known that refined specifications obtained from two bisimulation equivalent specifications are not guaranteed to be bisimulation equivalent. Unfortunately, it is same in incremental design methods. This article clarifies a pair of sufficient conditions of specifications and refinment in incremental design method to preserve bisimulation equivalence after refinement.

In Section 2, some definitions and notations used in this article are given. The language introduced in [IYK90] is adopted in this article. It is reviewed in Section 3. Action refinement and an equivalence called "*differential equivalence*" in [Itoh90] is reviewed in Section 4. Section 5 proposes the conditions for specifications and refinment. And we prove that bisimulation equivalence is preserved after refinement on the proposed conditions.

2 Notation and labeled transition system

This section explains the notation and definitions that are used in this article. Throughout the article, the end of a definition, theorem, etc. is marked by '▮', and the end of a proof is marked by '■'.

2.1 Basic notation

Notation 1 (Sets and Functions) Let A and B be sets, $a \in A$ and $b \in B$.

1) The *cardinality* of A is denoted by $\sharp(A)$.
$A \setminus B = \{a \mid a \in A \wedge a \notin B\}$. $A \setminus B$ is called the complement of B relative to A.

2) $\langle a, b\rangle$ denotes the ordered pair of a and b. $\langle A, B\rangle$ denotes the set of ordered pair.

3) $(A — B)$ denotes a set of all functions from A to B.

4) The set of natural numbers is denoted by "ω". ▮

Notation 2 Let A be a set, σ be a sequence in A^* and $i \in \omega$,

1) $\beta(i, \sigma)$ denotes the i-th element of σ.

2) $length(\sigma)$ denotes the length of σ.

3) $PrefixSet(\sigma) \overset{def}{=} \{\phi \in A^* \mid \exists\theta \in A^*[\phi\theta = \sigma]\}$.

4) $prefix(i, \sigma)$ denotes the prefix sequence of σ that length is i, i.e. $(prefix(i, \sigma) \in PrefixSet(\sigma)) \wedge (length(prefix(i, \sigma)) = i)$. ▮

Notation 3 1) "$\vec{a}$" is an abbreviation for sequence $a_1 \ldots a_n$.

2) Let a and b be elements of some set and $\mathcal{E}$ be an expression of the set.
$a^{\mathcal{E}\langle\langle i\rangle}$ denotes the i-th occurrence of a in $\mathcal{E}$.
$\mathcal{E}[b/a]$ denotes the expression obtained from $\mathcal{E}$ by substituting b for all occurrences of a. ▮

2.2 Labeled transition system

Definition 2.1 A *labeled transition system* TS is defined as a 4-tuple $TS = \langle S, \mathcal{A}, T, s_0 \rangle$, where:

- S is a (countable) non-empty set of states,
- $\mathcal{A}$ is a (countable) set of labels,
- $T = \{-\mu\rightarrow \ \subseteq S \times \mathcal{A} \times S \mid a \in \mathcal{A}\}$
  is a relation called the transition relation,
- $s_0 \in S$ is an initial state of TS. ▮

We sometimes use LTS as an abbreviation for labeled transition system.

Next, we introduce several formal notations for labeled transition systems.

Definition 2.2 Let B, B_i, C be labeled transition systems, $\mu, \mu_i \in \mathcal{A}$ and $\sigma \in \mathcal{A}^*$,

- $B{-}\mu_1 ... \mu_n \rightarrow_{\mathcal{A}} C$
  if $\exists B_i (0 \leq i \leq n)$
  $[B = B_0{-}\mu_1 \rightarrow B_1{-}\mu_2 \rightarrow ... {-}\mu_n \rightarrow B_n = C]$.
- $B{-}\mu_1 ... \mu_n \rightarrow_{\mathcal{A}}$ if $\exists C[B{-}\mu_1 ... \rightarrow C]$.
- $out_{\mathcal{A}}(B) \overset{def}{=} \{\mu \in \mathcal{A} \mid B{-}\mu\rightarrow\}$.
- $D_{\mathcal{A}}(B) \overset{def}{=} \{C \mid \exists \mu \in \mathcal{A}[B{-}\mu \rightarrow C]\}$.
- $S(B, \sigma)_{\mathcal{A}} \overset{def}{=} \{\mu \in \mathcal{A} \mid \exists C[B{-}\sigma \rightarrow C{-}\mu \rightarrow]\}$.
- $AS(B, \sigma)_{\mathcal{A}} \overset{def}{=} \{C \mid B{-}\sigma \rightarrow C\}$.
- $Tr_{\mathcal{A}}(B) \overset{def}{=} \{\sigma \in \mathcal{A}^* \mid B{-}\sigma\rightarrow\}$.
- $Path_{Act}(B) \overset{def}{=} \{p \mid (p \in Tr(B))$
  $\wedge(\forall q \in Tr(B)[p \in prefix(q) \Rightarrow q = p])\}$. ▮

Definition 2.3 *Act* is a (countable) set of observable actions. ▮

Usually, *Act* is used as the label set of LTS. But in discussion of refinement, $\langle Act, \omega \rangle$ is used as the label set. Therefor all notations in Definition 2.2 are defined with subscript indicating a label set. When $\mathcal{A}$ is apparent from context or not important, it will be often omitted.

3 A language that features a synthesis operator

In this section, we introduce the language $\mathcal{LS}$ discussed in this article. This language is defined in Subsection 3.1, and some of its properties are explained in Subsection 3.2.

3.1 Definition of $\mathcal{LS}$

$\mathcal{LS}$ has been defined in [IYK90], and we repeat its definition here.

Definition 3.1 Let $\mathcal{A}$ be a set, then $\mathcal{LS}_{\mathcal{A}}$ is the least set in which every terms is constructed from following only.

$\mathbf{stop} \in \mathcal{LS}_{\mathcal{A}}$.

$\mathbf{exit} \in \mathcal{LS}_{\mathcal{A}}$.

$B \in \mathcal{LS}_{\mathcal{A}}, \mu \in \mathcal{A} \Rightarrow \mu; B \in \mathcal{LS}_{\mathcal{A}}$.

$B_1 \in \mathcal{LS}_{\mathcal{A}}, B_2 \in \mathcal{LS}_{\mathcal{A}} \Rightarrow B_1[]B_2 \in \mathcal{LS}_{\mathcal{A}}$.

$B_1 \in \mathcal{LS}_{\mathcal{A}}, B_2 \in \mathcal{LS}_{\mathcal{A}}, G \subseteq \mathcal{A}$
$\Rightarrow B_1|[G]|B_2 \in \mathcal{LS}_{\mathcal{A}}$.

$B_1 \in \mathcal{LS}_{\mathcal{A}}, B_2 \in \mathcal{LS}_{\mathcal{A}} \Rightarrow B_1 \langle + \rangle B_2 \in \mathcal{LS}_{\mathcal{A}}$. ▮

Elements of $\mathcal{LS}_{\mathcal{A}}$ are called statements.

We will often abbreviate $\mathcal{LS}_{\mathcal{A}}$ to $\mathcal{LS}$, when $\mathcal{A}$ is apparent from context or not important.

To treat action refinement from an action to a statement, *action prefix* should be replaced by *sequential composition*. But in this article, only action refinement from an action to an action sequence is treated, so we do not replace.

Definition 3.2 Axioms of $\mathcal{LS}_{\mathcal{A}}$ combinators: $\mathcal{D}_m$

stop none.

exit $\mathbf{exit} \ {-}\delta \rightarrow \mathbf{stop} \ (\delta \notin Act)$.

$\mu; B$ $\quad \mu \in \mathcal{A} \vdash \mu; B{-}\mu \rightarrow B$.

$B_1[]B_2$ $\quad B_1{-}\mu \rightarrow B_1', out(B_1) \cap out(B_2) = \emptyset$
$\vdash B_1[]B_2{-}\mu \rightarrow B_1'$.
$B_2{-}\mu \rightarrow B_2', out(B_1) \cap out(B_2) = \emptyset$
$\vdash B_1[]B_2{-}\mu \rightarrow B_2'$.

$B_1|[G]|B_2$ $\quad B_1{-}\mu \rightarrow B_1', B_2{-}\mu \rightarrow B_2'$,
$out(B_1) \cap out(B_2) \subseteq G, \mu \in G$
$\vdash B_1|[G]|B_2{-}\mu \rightarrow B_1'|[G]|B_2'$.
$B_1{-}\mu \rightarrow B_1', out(B_1) \cap out(B_2) \subseteq G$,
$\mu \notin G$
$\vdash B_1|[G]|B_2{-}\mu \rightarrow B_1'|[G]|B_2$.
$B_2{-}\mu \rightarrow B_2', out(B_1) \cap out(B_2) \subseteq G$,
$\mu \notin G$
$\vdash B_1|[G]|B_2{-}\mu \rightarrow B_1|[G]|B_2'$.

$B_1 \langle + \rangle B_2$ $\quad B_1{-}\mu \rightarrow B_1', B_2{-}\mu \rightarrow B_2'$
$\vdash B_1 \langle + \rangle B_2{-}\mu \rightarrow B_1' \langle + \rangle B_2'$.
$B_1{-}\mu \rightarrow B_1', \mu \notin out(B2)$
$\vdash B_1 \langle + \rangle B_2{-}\mu \rightarrow B_1'$.
$B_2{-}\mu \rightarrow B_2', \mu \notin out(B1)$
$\vdash B_1 \langle + \rangle B_2{-}\mu \rightarrow B_2'$. ▮

Definition 3.3 $\mathcal{T}_{\mathcal{D}_m}$ is the smallest set that includes all transition rules derived from $\mathcal{D}_m$. ▮

Definition 3.4 $\langle \mathcal{LS}_{\mathcal{A}}, \mathcal{A}, \mathcal{T}_{\mathcal{D}_m}, S \rangle$ is called the labeled transition system corresponding to $S \in \mathcal{LS}_{\mathcal{A}}$ and denoted by $lts(S)$. ▮

In this article, we use "*strong equivalence*" in [Miln89]. The symbol "$\sim$" denotes strong equivalence. In this article we rename "strong equivalence" to "*strong bisimulation equivalence*" or merely "*bisimulation equivalence*".

Definition 3.5 $\mathcal{S}^{\mathcal{D}}{}_{\mathcal{A}}$ is the smallest subset of $\mathcal{LS}_{\mathcal{A}}$ that includes every statement S which satisfy following conditions.

For all $C \in D(S)$,

$$\text{if } C \equiv A_1[]A_2, \text{ then } out(A_1) \cap out(A_2) = \emptyset.$$

$$\text{if } C \equiv A_1|[G]|A_2, \text{ then } out(A_1) \cap out(A_2) \subseteq G. \text{ ▮}$$

3.2 Some properties of $\mathcal{LS}$

In this subsection, characteristic points and properties of $\mathcal{LS}$ are explained.

Proposition 1 If S is in $\mathcal{S}^{\mathcal{D}}{}_{\mathcal{A}}$ and σ is in $\mathcal{A}^*$, then $\sharp(AS_{\mathcal{A}}(S, \sigma)) \leq 1$. ▮

Proof. Omitted. ■

Definition 3.6 Let A be a statement and σ be in $Tr(A)$. The only element in $AS(A, \sigma)$ is denoted by $NAS(A, \sigma)$. ▮

Corollary 1 If A and B are statements in $\mathcal{S}^{\mathcal{D}}{}_{\mathcal{A}}$, then $A \sim B \Leftrightarrow Tr_{\mathcal{A}}(A) = Tr_{\mathcal{A}}(B)$. ▮

4 Action refinement in $\mathcal{LS}_{Act}$ and differential equivalence

In this section, we introduce refinement functions on $\mathcal{LS}_{Act}$. In subsection 4.1, we state our aim and mention some problems concerning action refinement in $\mathcal{LS}_{Act}$. In subsection 4.2, refinement functions on $\mathcal{LS}_{Act}$ and *differential equivalence* "$\sim_d$" are introduced. "differential equivalence" is one of the conditions on that a pair of pre-refined statements are refined to a pair of statements which are bisimulation equivalent.

4.1 The objective and problems in refining $\mathcal{LS}_{Act}$

We have been investigating incremental design methodology [IIKTS91]. The advantage of incremental design methodology is that new functions can be added easily after the systems have been put into use. Therefore throughout life cycle, programs can be extended. Such extend-ability is necessary to cope with the diverse requirements that are caused by increasing the number of users.

Computer systems, especially distributed systems are becoming larger. Large systems are usually designed step by step. At first they are designed abstractly, then stepwise designed in detailed. The increase in complexity of specifications makes specification modification more difficult. The motivation for our study of action refinement in $\mathcal{LS}_{Act}$ is the need to create a good design methodology for large systems that supports the addition of new functions after the systems has been put into use.

At the beginning, the relationship between refinement and addition of new functions is considered. A fundamental requirement is to preserve equivalence. So we investigated what pair of specifications preserve bisimulation equivalence by restricting the method of refining statements. Conditions for specifications and conditions for refinement depend on each other. This article takes up this problem and a pair of sufficient conditions of specifications and refinement are shown in Section 5.

$$\begin{tikzcd}
\text{What relation ?} \arrow[r, "-"] & \mathit{Abstract\ Spec}_1 \arrow[rr, mapsto] \arrow[dd, equal] & & \mathit{Refined\ Spec}_1 \arrow[dd, phantom, "\wr"] \\
& & \text{What refinement ?} \arrow[u] \arrow[d] & \\
& \mathit{Abstract\ Spec}_2 \arrow[rr, mapsto] & & \mathit{Refined\ Spec}_2
\end{tikzcd}$$

Figure 1: The issue investigated in this article.

Next, two problems in refining statements are listed. First example is a problem derived from action atomicity [Itoh90] [Acet92].

Example 1 Let $S_1 \equiv (a; b; \textbf{stop})[](b; a; \textbf{stop})$ and $S_2 \equiv (a; \textbf{stop})|[]|(b; \textbf{stop})$.

S_1 and S_2 are bisimulation equivalent.

Suppose that refinement by which action "a" is replaced with "$a_1; a_2$" and "b" with "$b_1; b_2$" is applied to S_1 and S_2.

S_1 is refined to

$(a_1; a_2; b_1; b_2; \mathbf{stop})[](b_1; b_2; a_1; a_2; \mathbf{stop})$

and S_2 is refined to $(a_1; a_2; \mathbf{stop})|[]|(b_1; b_2; \mathbf{stop})$.

After refinement is applied to these statements, they are not bisimulation equivalent. Refinement does not preserve bisimulation equivalence relation because actions are considered as atomic, instantaneous.

The next example occurs only in $\mathcal{LS}_{Act}$.

Example 2 Let $S_1 \equiv (a;\mathbf{stop})[](b;\mathbf{stop})$, $S_2 \equiv (a;\mathbf{stop})|[]|(b;\mathbf{stop})$. And let refinement by which action "a" is replaceed with "$a_1;a_2$" and "b" with "$a_1;b_2$" be applied to S_1 and S_2.

S_1 is refined to $(a_1; a_2; \mathbf{stop})[](a_1; b_2; \mathbf{stop})$ and S_2 is refined to $(a_1; a_2; \mathbf{stop})|[]|(a_1; b_2; \mathbf{stop})$.

Concerning refined S_1, the axiom of choice operator in Definition 3.3 cannot be applied to S_1 because $out(a_1; a_2; \mathbf{stop}) \cap out(a_1; b_2; \mathbf{stop}) \neq \emptyset$. So there is no axiom that can be applied to refined S_1. Therefore refined S_1 cannot translate.

Concerning refined S_2, the axiom of parallel operator in Definition 3.3 cannot be applied to S_2 because $out(a_1; a_2; \mathbf{stop}) \cap out(a_1; b_2; \mathbf{stop}) \not\subseteq \emptyset$.

So the LTS of a atatement obtaind by some refinement may be impossible to evolve into final state, although the its pre-refined statement's LTS evolves into final state without fail.

Taking these examples into consideration, we define action refinement in $\mathcal{LS}_{Act}$.

4.2 Refinement function of $\mathcal{LS}_{Act}$

Definition 4.1 Let S be a statement in $\mathcal{LS}_{Act}$ and $\gamma \in (Act \rightarrow Act^*)$, $Refin[\gamma]$ is in $(\mathcal{LS}_{Act} \rightarrow \mathcal{LS}_{Act})$ such that

$Refin[\gamma](S) = S[\widehat{\gamma(a)}/a]$ for each action a occurs in S, where

$\widehat{\gamma(a)} = \beta(1, \gamma(a)); ...; \beta(length(\gamma(a)), \gamma(a))$. ▮

Definition 4.2 $NumOc^S(a, A)$ is the number of a's occurrence in statement A. $NumOc^L(a, T)$ is the number of a's occurrence in LTS T. ▮

Next we introduce "*differential equivalence*" that is defined in [Itoh90]. This equivalence comes from the idea that all actions are not atomic, but they have a start and end. In differential equivalence, each action is divided into two parts indicating its start and end.

Definition 4.3 Let S, S_1, S_2 be a statement in $\mathcal{LS}_{Act}$.

$d(S)$ denotes statement in $\mathcal{LS}_{\langle Act, \{\mathbf{S},\mathbf{E}\}\rangle}$ that each occurrence a of all actions in A is replaced with $\langle a, \mathbf{S}\rangle; \langle a, \mathbf{E}\rangle$.

S_1 and S_2 are differential equivalent iff $d(S_1) \sim d(S_2)$.

$A \sim_d B$ denotes S_1 and S_2 are differential equivalent. ▮

Notion 1 Occurrences of $\langle a, \mathbf{S}\rangle$ and $\langle a, \mathbf{E}\rangle$ partioned from the same element are said to correspond. ▮

Proposition 2 Let S_1, S_2 be statements of $\mathcal{LS}_{Act}$.

If $S_1 \sim_d S_2$ then $S_1 \sim S_2$. ▮

Proof. Omitted(see [Itoh90]). ■

5 The conditions of refinement that preserve bisimulation equivalence

In this section, a pair of sufficient conditions to preserve bisimulation equivalence after refinement is presented. The conditions consist of a condition for pair of statements and a condition on refinement functions.

Refinement is divided into two step: the first step is only replacing each action with an sequence in $\langle A, \omega\rangle^*$; the second step relabels each $\langle a, i\rangle$ in $\langle A, \omega\rangle$ to an action in Act.

In subsection 5.1, it is shown that if two statements are differential equivalent, action-partitioned statements are also bisimulation equivalent.

In subsection 5.2, we define a set of refinement functions $\mathcal{FS}_{Act}(\mathcal{SP})$ for refining statements that preserve bisimulation equivalence, where $\mathcal{SP}$ is a subset of $\mathcal{S^D}_{Act}$. And it is shown that if $S_1, S_2 \in \mathcal{SP}$ and $S_1 \sim_d S_2$, then S_1 and S_2 are refined to statements that are bisimulation equivalent by any function in $\mathcal{FS}_{Act}(\mathcal{SP})$.

5.1 First step: partitioning actions

In this section, replacement of each action in statements with a sequence in $\langle Act, \omega\rangle^*$ is shown.

First, functions in $(Act - \langle Act, \omega\rangle^*)$ is defined.

Definition 5.1 Let A be in $\mathcal{LS}$ and Γ_P be a function in $(Act - \omega \setminus \{1\})$.

$Part[\Gamma_P]$ is a function in $(\mathcal{LS}_{Act} \rightarrow \mathcal{LS}_{\langle Act,\omega\rangle})$ as follows.

In $Part[\Gamma_P](A)$, each occurrence of all actions in A is replaced with expressions such that

if $\Gamma_P(a) = 0$, action a is replaced with $\langle a, 0\rangle$,

if $\Gamma_P(a) \geq 2$, action a is replaced with $\langle a, 1\rangle; ...; \langle a, \Gamma_P(a)\rangle$. ▮

Definition 5.2 Let $\mathcal{E}_1$ be a statement and $\mathcal{E}_2 = Part[\Gamma_P](\mathcal{E}_1)$.

Occurrences of $\langle a, i\rangle$ in $\mathcal{E}_2$ are called

$Part[\Gamma_P]$-*descendant*$(a^{\mathcal{E}_1\langle\!\langle k\rangle\!\rangle})$ in $\mathcal{E}_2$ if their occurrences correspond to the occurrence $a^{\mathcal{E}_1\langle\!\langle k\rangle\!\rangle}$ in $\mathcal{E}_1$. ▮

An important operator for labeled transition systems is defined. In the rest of this subsection, all lemmas and propositions concern this operator.

Definition 5.3 Let $\langle a, i\rangle$ be a transition label in $lts(S)$, S be in $\mathcal{LS}_{\langle Act,\omega\rangle}$ and

$1 \leq k \leq NumOc^L(\langle a, i\rangle, S)$.

$CorOc(\langle a, i\rangle, k, S)$ is a smallest set of natural numbers such that :

if k-th occurrence of $\langle a, i\rangle$ in $lts(S)$ indicates the transition by m-th occurrence of $\langle a, i\rangle$ in A, $m \in CorOc(\langle a, i\rangle, k, S)$. ▮

By synthesis operator, several actions are synthesized to one transition in labeled transition systems. Therefore $CorOc$ is defined as a set.

Definition 5.4 Let S be a statement of $\mathcal{S}^{\mathcal{D}}{}_{\langle Act,\omega\rangle}$. $op[\langle a_1, i_1\rangle, ..., \langle a_n, i_n\rangle](S)$ is defined as follows[1],

i) $op[\langle a_1, 2\rangle, ..., \langle a_n, 2\rangle](S) \equiv lts(Part[\Gamma_P](S))$, where $\Gamma_P \in (Act \rightarrow \omega \setminus \{1\})$ such that

$$\Gamma_P(x) = \begin{cases} 2 & \text{if } x = a_m (1 \leq m \leq n), \\ 0 & \text{otherwise.} \end{cases}$$

ii) Let $i_j \geq 2 (1 \leq j \leq n)$.

$op[\langle a_1, i_1\rangle, ..., \langle a_s, i_s + 1\rangle, ..., \langle a_n, i_n\rangle](S)$ be determined from every occurrences of $\langle a_s, i_s\rangle$ in $op[\langle a_1, i_1\rangle, ..., \langle a_s, i_s\rangle, ..., \langle a_n, i_n\rangle](S)$ as follows:

Let $1 \leq j \leq NumOc^L(\langle a_s, i_s\rangle, op[\langle \overrightarrow{a,i}\rangle](S))$.

For the j-th occurrence of $\langle a_s, i_s\rangle$, there is a σ in $\mathcal{LS}_{\langle Act,\omega\rangle}$ such that

$op[\langle a_1, i_1\rangle, ..., \langle a_s, i_s\rangle, ..., \langle a_n, i_n\rangle](S)$
$\quad -\sigma \rightarrow (\langle a_s, i_s\rangle^{op[\langle \overrightarrow{a,i}\rangle](S)\langle\!\langle j\rangle\!\rangle}; Y) \mid Z$,

where "|" denotes the branch of LTS and ";" denotes concatenation of LTS.

Let $W \equiv (\langle a_s, i_s\rangle^{op[\langle \overrightarrow{a,i}\rangle](S)\langle\!\langle j\rangle\!\rangle}; Y) \mid Z$.

To obtain $op[\langle a_1, i_1\rangle, .., \langle a_s, i_s + 1\rangle, .., \langle a_n, i_n\rangle](S)$, W is replaced with W' that is constructed as follow.

Let $Z' \equiv Z[\langle a_s, i_s + 1\rangle / \langle a_s, i_s\rangle^{op[\langle \overrightarrow{a,i}\rangle](S)\langle\!\langle k\rangle\!\rangle}]$, for all k that hold $CorOc(\langle a_s, i_s\rangle, k, S) \subseteq CorOc(\langle a_s, i_s\rangle. j, S)$.

$W' \equiv Z \mid (\langle a_s, i_s\rangle^{op[\langle \overrightarrow{a,i}\rangle](S)\langle\!\langle j\rangle\!\rangle}; (Z' \mid \langle a_s, i_s + 1\rangle; Y))$. ▮

Lemma 1 Let $a_i \in Act$ $(1 \leq i \leq j)$ and A, $B \in \mathcal{S}^{\mathcal{D}}{}_{Act}$.

$A \sim_d B \Rightarrow$

$op[\langle a_1, 2\rangle ... \langle a_j, 2\rangle](A) \sim op[\langle a_1, 2\rangle ... \langle a_j, 2\rangle](B)$. ▮

Proof. This is obvious from Definition 5.4. ▮

Lemma 2 Let $n_i > 1$ $(1 \leq i \leq j)$ and A, $B \in \mathcal{S}^{\mathcal{D}}{}_{Act}$.

For all $1 \leq s \leq j$

$op[\langle a_1, n_1\rangle, ..., \langle a_j, n_j\rangle](A)$

$\sim op[\langle a_1, n_1\rangle, ..., \langle a_j, n_j\rangle](B)$

$\Rightarrow$

$op[\langle a_1, n_1\rangle, ..., \langle a_s, n_s + 1\rangle ..., \langle a_j, n_j\rangle](A)$

$\sim op[\langle a_1, n_1\rangle, ..., \langle a_s, n_s + 1\rangle ..., \langle a_j, n_j\rangle](B)$. ▮

Proof.

From Definition 5.4, $op[...\langle a, i\rangle ...](X)$ is such that

$op[\langle a, i\rangle](X) - \sigma \rightarrow W_X$

and $W_X \equiv Z_X \mid \langle a, i\rangle^{op[\langle \overrightarrow{a,i}\rangle](X)\langle\!\langle k_X\rangle\!\rangle} (X = A, B)$.

In $op[\langle a, i + 1\rangle](X)$, W_X is replaced with W' as described below.

$W'_X \equiv Z_X \mid \langle a, i\rangle; (Z'_X \mid \langle a, i + 1\rangle; Y_X)$.

$Z'_X \equiv Z_X[\langle a, i + 1\rangle / \langle a, i\rangle^{op[\langle \overrightarrow{a,i}\rangle](X)\langle\!\langle k_X\rangle\!\rangle}]$.

From the hypothesis, $W_A \sim W_B$.

So $Z_A \sim Z_B$ and $Y_A \sim Y_B$.

When $i \geq 2$, from $A, B \in \mathcal{S}^{\mathcal{D}}{}_{Act}$, there is a $\langle a, i\rangle$ in each $Path_{\langle Act,\omega\rangle}(Z_X)$ and first $\langle a, i\rangle$'s occurrences in each path in $Path(Z_X)_{\langle Act,\omega\rangle}(Z_X)$. correspond to $a^{X\langle\!\langle m\rangle\!\rangle}$ in X, where $m \in CorOc(\langle a, i\rangle, k, X)$. Moreover only these occurrences correspond to $a^{X\langle\!\langle m\rangle\!\rangle}$ in X for all $m \in CorOc(\langle a, i\rangle, k, X)$.

So $Z'_A \sim Z'_B$.

Therefore

$op[\langle a_1, n_1\rangle, ... \langle a_j, n_j\rangle](A)$

$\sim op[\langle a_1, n_1\rangle, ... \langle a_j, n_j\rangle](B)$

[1] We sometimes use the expression $op[\langle \overrightarrow{a,i}\rangle](S)$ as an abbreviation for $op[\langle a_1, i_1\rangle, ..., \langle a_n, i_n\rangle](S)$.

$\Rightarrow$
$op[\langle a_1, n_1\rangle, ..., \langle a_s, n_s + 1\rangle ..., \langle a_j, n_j\rangle](A)$
$\sim op[\langle a_1, n_1\rangle, ..., \langle a_s, n_s + 1\rangle ..., \langle a_j, n_j\rangle](B)$. ∎

From the above lemmas, following proposition is proved.

Proposition 3 *Let A and B be in $\mathcal{S}^{\mathcal{D}}{}_{Act}$.*
$A \sim_d B \Rightarrow$
$op[\langle a_1, n_1\rangle, ..., \langle a_j, n_j\rangle](A)$
$\sim op[\langle a_1, n_1\rangle, ..., \langle a_j, n_j\rangle](B)$. ∎

The rest of this subsection, relationship between "*op*-operator" and "$Part[\Gamma_P]$" is stated.

Lemma 3 Let S be a statement in $\mathcal{S}^{\mathcal{D}}{}_{Act}$,
$\Gamma_P \in (Act \rightarrow \omega)$, and $i \in \omega$.
If in $lts(d(S))$, transitions that occur just after any occurrence of $\langle a, \mathbf{S}\rangle$ that corresponds to $a^{S\langle\langle k\rangle\rangle}$ in S are only transitions by $\langle a, \mathbf{E}\rangle$, then in $lts(S_{\langle Act,\omega\rangle})$ transitions that occur just after any occurrence of $\langle a, i\rangle$ that correspond to a in S are only transitions by
$\langle a, i + 1\rangle$. ∎

Proof.

In $lts(Part[\Gamma_P](S))$, a transition by $\langle b, j\rangle$ occurs after transitions by $\langle a, i\rangle$
$(1 \leq i < \Gamma_P(a))$ iff there are occurrences of a and b that both within the scope of some "$|[]|$" in S. Although applying "$Part[\Gamma_P]$" to S, in each scope of "$|[]|$" in $lts(Part[\Gamma_P](S))$ there is no $Part[\Gamma_P]$*-descendant* of actions that is out of scope of the "$|[]|$" in S. So this lemma holds. ∎

When statements are refined, some attention should pay to the influence of the synthesis operator.

Example 3
Let $S \equiv (a; \mathbf{stop})\langle\!+\!\rangle((b; \mathbf{stop})|[]|(c; \mathbf{stop}))$.
In $lts(d(S))$ there is no transition except by $\langle a . \mathbf{E}\rangle$ after transition by $\langle a, \mathbf{S}\rangle$.
Suppose $\gamma \in (Act \rightarrow Act^+)$ is as follows,

$$Refin[\gamma](x) = \begin{cases} d_1 d_2 & \text{if } x = a, \\ d_1 d_3 & \text{if } x = b, \\ e_1 e_2 & \text{if } x = c. \end{cases}$$

By synthesis operator, there are transitions by d_2, d_3 and e_1 just after d_1. Among them only d_2 is come out from a by the refinement. It is caused because $Refin[\gamma](a)$ and $Refin[\gamma](b)$ have same prefix. On applying $Part[\Gamma_P]$ ($\Gamma_P \in (Act \rightarrow \omega)$), if $a \neq b$ then $Part[\Gamma_P](a)$ and $Part[\Gamma_P](b)$ have not same prefix. So this cannot be a counter-example for Lemma 3.

Proposition 4 Let S be a statement in $\mathcal{LS}_{Act}$, $i_j \geq 2$ $(1 \leq j \leq n)$ and Γ_P be a function in $(Act \rightarrow \omega \setminus \{1\})$ such that

$$\Gamma_P(x) = \begin{cases} i_j & \text{if } x = a_j \ (1 \leq j \leq n), \\ 0 & \text{otherwise.} \end{cases}$$

Then
$op[\langle a_1, i_1\rangle, ..., \langle a_n, i_n\rangle](S) \sim lts(Part[\Gamma_P](S))$. ∎
Proof. This proof is Omitted. ∎

From the above two propositions, $S_1 \sim_d S_2 \Rightarrow Part[\Gamma_P](S_1) \sim Part[\Gamma_P](S_2))$ is derived.

5.2 Second step: relabel action fragments, and the theorem

In this subsection relabeling from $\langle a, i\rangle$ to a is discussed.

Definition 5.5 Let S be a statement in $\mathcal{LS}_{\langle Act,\omega\rangle}$ and Γ_R be a function in $(\langle Act, \omega\rangle \rightarrow Act)$.
$Relab[\Gamma_R]$ is a function in $(\mathcal{LS}_{\langle Act,\omega\rangle} \rightarrow \mathcal{LS}_{Act})$ such that $Relab[\Gamma_R](S) \equiv S[\Gamma_R(\langle a, i\rangle)/\langle a, i\rangle]$ for all $\langle a, i\rangle$. ∎

Definition 5.6
Let $\mathcal{SP}$ be a subset of $\mathcal{S}^{\mathcal{D}}{}_{\langle Act,\omega\rangle}$. $\mathcal{FS}_{\langle Act,\omega\rangle}(\mathcal{SP})$ is the smallest subset of $(\langle Act, \omega\rangle \rightarrow Act)$ that includes every function Γ_R which satisfys the below.
For all $A \in \bigcup_{S \in \mathcal{SP}} D_{\langle Act,\omega\rangle}(S)$

i) if $A \equiv A_1[]A_2$. $\Gamma_R[out(A_1)] \cap \Gamma_R[out(A_2)] = \emptyset$.

ii) if $A \equiv A_1|[G]|A_2$, $\forall\langle a, i\rangle \in G[\forall\langle b, j\rangle \in \langle Act, \omega\rangle[a \neq b \Rightarrow \Gamma_R(\langle a, i\rangle) \neq \Gamma_R(\langle a, j\rangle)]]$. ∎

Let σ be a sequence of elements in the domain of function f. $f[\sigma]$ is a sequence of elements in the range of f such that $f[\sigma] = f(\beta(1, \sigma))...f(\beta(length(\sigma), \sigma))$.

Lemma 4 If S is a statement in $\mathcal{SP}_{\langle Act,\omega\rangle}$ and $\Gamma_R \in \mathcal{FS}_{\langle Act,\omega\rangle}$, then $Tr(Relab[\Gamma_R](S)) = \{\Gamma_R[\sigma] \mid \sigma \in Tr(S)\}$. ∎

Proof. We prove this by induction on structure of statements.

1) If S is **stop** or **exit** , the proof is obvious. Assume that A_1 and A_2 hold $Tr(Relab[\Gamma_R](X))$
$= \{\Gamma_R[\sigma] \mid \sigma \in Tr(X)\}$ $(X = A_1, A_2)$.

2) We omit cases ";", "[]", "⟨+⟩" and show "$|[G]|$". $\sigma \in Tr(A_1|[G]|A_2)$ is a sequence as below. For $A_1|[G]|A_2 \in \mathcal{S}^{\mathcal{D}}$, $lts(A_1|[G]|A_2)$ is translated by

$\langle a, l\rangle$, when A_1 can be translated and A_2 cannot be translated by it.

$\langle a, l\rangle$, when A_2 can be translated and A_1 cannot be translated by it.

$\langle a, l\rangle$ when both A_1 and A_2 can be translated synchronously by it.

So transition sequences of A_1 and A_2 can be extracted from σ. These sequences are denoted by $proj(\sigma, A_k)$. $proj(\sigma, A_k) \in Tr(A_k)(k = 1, 2)$.
This lemma is proven by showing below ,
$Tr(Relab[\Gamma_R](A_1|[G]|A_2))$
$\subseteq \{\Gamma_R[\sigma] \mid \sigma \in Tr(A_1|[G]|A_2)\}$
and $\{\Gamma_R[\sigma] \mid \sigma \in Tr(A_1|[G]|A_2)\}$
$\subseteq Tr(Relab[\Gamma_R](A_1|[G]|A_2))$.
Rest of this proof is omitted. ■

Proposition 5 If S_1, $S_2 \in \mathcal{SP}$, $S_1 \sim S_2$. $\Gamma_R \in$ and $\mathcal{FS}_{\langle Act,\omega\rangle}$, then $Relab[\Gamma_R](S_1) \sim Relab[\Gamma_R](S_2)$. ■

- **Proof.**
From Corollary 1, $S_1 \sim S_2 \Leftrightarrow Tr(S_1) = Tr(S_2)$.
From Lemma 4, $Tr(Relab[\Gamma_R](S_i))$
$= \{\Gamma_R[\sigma] \mid \sigma \in Tr(S_i)\}(i = 1, 2)$.
So $Tr(Relab[\Gamma_R](S_1)) = Tr(Relab[\Gamma_R](S_2))$.
Then using Corollary 1 again,
$Relab[\Gamma_R](S_1) \sim Relab[\Gamma_R](S_2)$. ■

Definition 5.7 Let $\gamma \in (Act \rightarrow Act^*)$ and $\mathcal{SP}_{Act} \subseteq \mathcal{S}^{\mathcal{D}}$.

$\mathcal{FS}_{Act}(\mathcal{SP}_{Act}) = \{\gamma \mid \exists\Gamma_P \in (Act - \omega)[\exists\Gamma_R \in \mathcal{FS}_{\langle Act,\omega\rangle}(Part[\Gamma_P][\mathcal{SP}_{Act}])[\gamma = \Gamma_R \circ \Gamma_P]]\}$. ■

From above propositions, lemmas and a corollary, follow theorem are derived.

Theorem 1 Let S_1, S_2 be statements in $\mathcal{S}^{\mathcal{D}}{}_{Act}$ and $\mathcal{SP}$ be a subset of $\mathcal{S}^{\mathcal{D}}{}_{Act}$ that includes S_1 and S_2.
For any $\gamma \in \mathcal{FS}_{Act}(\mathcal{SP})$,
$S_1 \sim_d S_2 \Rightarrow Refin[\gamma](S_1) \sim Refin[\gamma](S_2)$. ■

With synthesis operator, turning points in LTSs of refined statements may differ from the positions just before corresponding actions in pre-refined ones. Turning points in LTSs of refined statements may occur in the middle of actions in pre-refined statements. But if a pair of pre-refined statements are *differential equivalent*, a pair of bisimulation equivalent statements are obtained by any $\gamma \in \mathcal{FS}_{\mathcal{SP}}$.

6 Conclusion

This article discusses foundations of two concepts of design: action refinement and specification synthesis, that are important for complex systems design.

Proposed is a pair of sufficient conditions for preserving bisimulation equivalence after refinement.

Acknowledgements

The many helpful discussions with Masaki Itoh, Eiichi Horita and Kenjiroh Yamanaka are gratefully acknowledged.

References

[Acet92] L. Aceto (1992), *Action refinement in process algebras*, Cambridge University Press.

[BSS87] E. Brinksma, G. Scollo, C. Steenbergen (1987). *LOTOS Specification, Their Implementations and Their Tests*, Protocol Specification, Testing and Verification, VI, (North Holland).

[Glab90] R. J. V. Glabbeek (1990), *Comparative Concurrency Semantics and Refinement of Actions*, Ph.D. thesis, the Free University of Amsterdam.

[IYK90] H. Ichikawa, Y. Yamanaka, J. Kato (1990), *Incremental Specification in LOTOS*, Protocol Specification, Testing and Verification, X, (North Holland).

[IIKTS91] H. Ichikawa, M. Itoh, J. Kato, et. al (1991), *SDE: Incremental Specification and Development of Communications Software*, IEEE Trans. on Computers 40, 4.

[Itoh90] M. Itoh (1990), *A Process Algebra Featuring Action Refinement* The Trans. of the IEICE, Vol.E-73, No.11.

[LOTOS] *ISO(1989). Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique based on the Temporal Ordering of Observational Behavior* International Standard ISO 8807 .

[Miln89] R. Milner (1989), *Communication and Concurrency*, Prentice Hall.