AN APPLICATION OF PARTIAL EVALUATION OF COMMUNICATING PROCESSES TO SYSTEM SECURITY

Masaki Murakami

1Department of Computer Science, Graduate School of Natural Science and Technology, Okayama University, 3-1-1, Tsushima-naka, Kitaku, Okayama, 700-0082, Japan
murakami@momo.cs.okayama-u.ac.jp

ABSTRACT

This paper presents a framework that extends a partial evaluation method for transformational programs to a method for reactive CSP processes. Temporal logic formulas are used to represent constraints on the sets of the sequences of communication actions executed by the processes. We present a set of simple rules for specializing processes with temporal formulas which contain $\chi$(next)-operators and/or $\Xi$(invariant)-operators. We show the soundness of the rules. Our partial evaluation method specializes reactive processes with the specifications of their environments. Furthermore, we present an example of an application of our partial evaluation method to improve the security of concurrent systems.

KEYWORDS

partial evaluation, CSP, reactive processes, temporal logic, Needham-Schroeder-Lowe protocol

1. INTRODUCTION

Partial evaluation is a method for optimizing programs using constraints on the input values to improve their efficiency. A number of results are reported in various languages[13]. In the common basic idea of conventional partial evaluation methods, we regard a program as a function from the domain of input values to the range of output values. Let $P_f$ be a program that is an implementation of a function $f(x, y)$. A typical partial evaluation is for example, to obtain a new program $P_{fa}$ from $P_f$ and an input value $a$, where $P_{fa}$ is the implementation of the function $f(a, y)$ (Figure 1). More general one is to obtain an implementation of the function $f_a(x)$ such that “for any $x$, if $p(x)$ is true then $f(x) = f_a(x)$”, from a function $f(x)$ and a constraint $p(x)$ on the input[5,6].

These methods are based on the transformational paradigm on which a program is an implementation of a function from the domain of input values to the range of output values.

DOI:10.5121/ijfcst.2012.2402
On the other hand, a program is considered based on the reactive paradigm in the area of concurrent computation[9]. A program is not a function from input values to output values on the reactive paradigm. It is a process which communicates with its environment during the computation. For example a process $P_1$ communicates with the environment by action $m_1$, then becomes the subsequent process $P_2$ which makes another action $m_2$ and so on as Figure 2. In this case, not only $m_1$ but also $m_2$ affects to the behavior of $P_2, P_3, \ldots$ and so on.

![Figure 2. A Reactive Process](image)

Thus we can expect to improve the efficiency of this program by partial evaluation not only with the constraint on the first action $m_1$ but also with the constraints on the values and the order of actions $m_2, m_3, \ldots$. A number of partial evaluation method for concurrent programs are reported [3,11,15,16,17,21,22]. However, these results do not consider the specialization of concurrent processes wrt the constraints for reactive environments of processes.

The purpose of this paper is to present a framework that extends a partial evaluation method for transformational programs to a method for reactive processes. We present a simple set of the rules for partial evaluation of reactive CSP[10,20] processes. We use temporal logic formulas[2] for the constraints on the environment of processes. Temporal formulas represent the constraints on the set of sequences of the possible communication actions of processes. The class of formulas used here is the set of temporal formulas which contain $X$(next)-operators and/or $G$(invariant)-operators.

We show the soundness of the rules. Our method makes possible to specialize a concurrent program with the specification of its environment to improve its efficiency. Furthermore, we present an example of an application of our partial evaluation method to improve the security of concurrent systems.

### 2. REACTIVE PROCESS AND ITS ENVIRONMENT

#### 2.1. Reactive Process

We adopt CSP (Communicating Sequential Processes) [10,20] notation to describe reactive processes in this paper.

A process is a function from an action to a process in the framework of CSP. An action is an input communication $c?x$ or an output communication $c!v$, where $c$ is a channel name and $x$ is a variable and $v$ is a term. When $c?x$ in process $P_1$ and $c!v$ in process $P_2$ are executed simultaneously, the value of $x$ becomes $v$. Thus $v$ is transferred from $P_2$ to $P_1$ using the channel $c$.

A process which maps a communication action $m_i$ to a process $P_i$ is denoted by guarded command like notation such as $(m_1 \rightarrow P_1 | \ldots | m_i \rightarrow P_i | \ldots)$ or $(m_i \rightarrow P_i)_{i \in I}$ for a set of indexes $I$ in short. We consider, for example, $(c?x \rightarrow P(x))$ as $(c?t_i \rightarrow P(t_i))_{i \in I}$ where the domain of $x$ is $\{t_i | i \in I\}$ as usual. The semantics of a process is defined with a failure set [10,20].
Definition 2.1  \( \text{traces}(P) \) is the set of sequences of actions that a process \( P \) can perform defined as follows.

\[
\text{traces}((m_i \rightarrow P_i)_{i \in I}) = \{ < > \} \cup \{ < m_i >^s \mid s \in \text{traces}(P_i), \ i \in I \}
\]

\( s_1 \wedge s_2 \) is the concatenation of a finite sequence \( s_1 \) and a finite (or an infinite) sequence \( s_2 \). Namely, if \( s_1 = < m_1^1, m_1^2, \ldots, m_1^k > \) and \( s_1 = < m_2^1, m_2^2, \ldots, m_2^h > \) (or \( s_1 = < m_2^1, m_2^2, \ldots > \)) then

\[
s_1 \wedge s_2 = < m_1^1, m_1^2, \ldots, m_1^k, m_2^1, m_2^2, \ldots, m_2^h > \quad \text{(or} < m_1^1, m_1^2, \ldots, m_1^k, m_2^1, m_2^2, \ldots > \text{)}
\]

\(< > \) is the empty sequence.

Definition 2.2  If \( P = (m_i \rightarrow P_i)_{i \in I} \) then:

\[
\text{failures}(P) = \{ (< m_i >^s, R) \mid (s, R) \in \text{failures}(P_i) \} \cup \{ (< >, R) \mid \forall i \in I, m_i \notin R \}
\]

For a substitution \( \theta \),

\[
\text{failures}(A\theta) = \text{failures}(F\theta) \quad \text{where} \quad A \overset{\text{def}}{=} F.
\]

A process which behaves like \( P_1 \) or \( P_2 \) nondeterministically is denoted as:

\[
P_1 \parallel P_2.
\]

It is impossible from the environment to control which of \( P_1 \) or \( P_2 \) is selected.

Definition 2.3

\[
\text{failures}(P_1 \parallel P_2) = \text{failures}(P_1) \cup \text{failures}(P_2).
\]

2.2. Environments of Reactive Processes

Let \( P \) and \( P' \) be processes. Consider that there is no difference between the sets of possible behaviors of processes \( P \) and \( P' \) for a given environment. In other words, \( P \) and \( P' \) cannot be distinguished under the environment. If \( P' \) can be executed more efficiently than \( P \), then we should adopt \( P' \) rather than \( P \) when it is executed in the environment.

For example, consider the following process that receives a finite list \( x \) of numbers, returns the maximum value in \( x \) and become itself again.

\[
P \overset{\text{def}}{=} (c_{in} \cdot ?x \rightarrow (c_{out} \cdot !\text{max}(x) \rightarrow P))
\]

If \( x \) is always sorted in descending order (that implies \( \text{max}(x) = \text{head}(x) \)) then it is impossible to distinguish \( P \) from the following process \( P' \) with their observable behaviors.

\[
P' \overset{\text{def}}{=} (c_{in} \cdot ?x \rightarrow (c_{out} \cdot !\text{head}(x) \rightarrow P))
\]

The environment of a reactive process \( P \) is the set of processes which communicate with \( P \). The constraints on the environments are derived from the specification of the environment processes that communicate with the target process. The specification of
environment processes can be given with temporal logic formulas\cite{2}. We use temporal operators such as $G$ and/or $X$ in formulas. (We do not consider formulas that contain $F$ (possible) operators.) The intuitive meanings of each operator are:

- $Xp$: $p$ holds at the next.
- $Gp$: $p$ always holds.

The truth value of a temporal formula is defined on a sequence which denotes the time axis. Usually, the truth values of formulas without temporal operators that represent the constraints on the input value $a$ are defined using the notion of state of a program which is decided from the values of input variables. However, as truth values of formulas without temporal formulas are not essential here, we avoid to discuss that in detail. We assume that for any formula $p$ without temporal operators, the set $\text{model}of(p)$ of sequences that make $p$ true on them is defined.

In this paper, a time axis is a trace of a process. Let $s$ be a finite sequence of communication actions $<m_1, m_2, \ldots, m_n>$ or an infinite sequence $<m_1, m_2, \ldots>$. We denote $\text{head}(m) = m_1$, and $\text{tail}_i(s) = <m_i, \ldots, m_n>$ (or $<m_i, \ldots>$). Note that $\text{tail}_1(s) = s$.

**Definition 2.4** Let $s$ be a sequence of communication actions.

1. If $p(x)$ be a predicate formula which does not contain temporal operators, and $x$ be a variable on the set of communication actions or a variable that occurs in communication actions.
   
   $$s \models p \iff s \in \text{modelof}(p).$$

2. $s \models Xp$ iff $\text{tail}_2(s) \models p$.

3. $s \models Gp$ iff $\forall i(1 \leq i) \text{tail}_i(s) \models p$.

We introduce the predicate $\text{done}(m)$. Intuitively, if $\text{done}(m)$ is true on a state then the action $m$ is “just finished” before reaching the state. For a substitution $\theta$ and a term $v$, we denote $c!(v\theta)$ as $m\theta$ if $m = c!v$. Similarly, we denote $c?(x\theta)$ as $m\theta$ if $m = c?x$ for a variable $x$.

**Definition 2.5** For some substitution $\theta$,

$$<m_0>^s \models X\text{done}(m) \iff m\theta = m_0$$

For example,

$$G(\text{done}(c?\text{pop}) \supset X\text{done}(c!\text{send}(0)))$$

means that if $c$ receives “pop” the it always sends “send(0)” immediately.

**Definition 2.6** Let $M$ be a set of communication actions.

$$M \models p \iff \forall s \in M, s \models p$$

For a process $P$ and a formula $p$,

$$P \models p \iff \forall t \in \text{traces}(P), t \models p$$

Namely $p$ is true of the set of sequences of communication actions such that $P$ executes with the environment.
3. PARTIAL EVALUATION OF COMMUNICATING PROCESSES

3.1. Set of Rules

Let $P$ be a process and $p$ be a temporal formula which represents a constraint for the environment of $P$. We denote the partial evaluation of process $P$ by $p$ as $\text{Part}(P, p)$. This section presents a method to obtain a new process which is equal to $\text{Part}(P, p)$.

Let $P_0$ be a new process name and let the definition of $P_0$ be:

$$P_0 \overset{\text{def}}{=} \text{Part}(P, p).$$

We rewrite the right hand side of this definition using the following rules.

- **Unfolding/Folding**
  
  For $P \overset{\text{def}}{=} F$,
  
  Unfolding $P\theta \Rightarrow F\theta$
  
  Folding $F\theta \Rightarrow P\theta$

  where $P\theta$ (or $F\theta$) is an instance of $P$ (or $F$ respectively) which is obtained by applying a substitution $\theta$.

- **Non-temporal logical rules**

  **Predicate rule**
  
  $\text{Part}(P, p) \Rightarrow P_p$
  
  where $p$ is a predicate formula without temporal operators, and $P_p$ is obtained with partial evaluation from $P$ and $p$ (by a conventional method).

  **done rule**
  
  $(m \rightarrow P) \Rightarrow (m \rightarrow \text{Part}(P, \text{done}(m)))$

  **$\land$ rule**
  
  $\text{Part}(P, p \land q) \Rightarrow \text{Part}(\text{Part}(P, p), q)$

  **$\land^+$ rule**
  
  $\text{Part}(\text{Part}(P, p), q) \Rightarrow \text{Part}(P, p \land q)$

  **$\supset$ rule**
  
  $\text{Part}(P, p) \Rightarrow \text{Part}(P, q)$ if $p \supset q$

- **Temporal rules**

  Let $P$ be a process which is equal to $(m_i \rightarrow P_i)_{i \in I}$.

  **$\times$ rule**
  
  $\text{Part}(P, Xp) \Rightarrow (m_i \rightarrow \text{Part}(P_i, p))_{i \in I}$

  **$G$ rule**
  
  $\text{Part}(P, Gp) \Rightarrow \text{Part}((m_i \rightarrow \text{Part}(P_i, Gp))_{i \in I}, p)$

  **Pruning rule**
  
  $\text{Part}(P, \times\text{done}(m_i\theta)) \Rightarrow (m_i\theta \rightarrow P_i')$

  for $k \in I$ if $m_i\rho \neq m_i\theta$ for any substitution $\rho$ and any $j(\in I) \neq k$

- **$\sqcap$ rule**

  $\text{Part}(P_1 \sqcap P_2, p) \Rightarrow \text{Part}(P_1, p) \sqcap \text{Part}(P_2, p)$
Termination rule

Definition 3.1 Let \( E_0 = \text{Part}(P, p) \). A finite sequence \( E_0, E_1, E_2, \ldots, E_n \) is a transformation sequence if \( E_i \) is obtained from \( E_{i-1} \) directly by applying one of the rules above.

If \( E_n \) no longer contains a sub-expression in the form of \( \text{Part}(Q, q) \), then the partial evaluation is completed.

3.2. Example

Let the definition of \( P \) be:
\[
P \overset{\text{def}}{=} (c_{in}\,?x \rightarrow (c_{out}\,!\text{max}(x) \rightarrow P))
\]
as Section 2.2. \( \text{Part}(P, Gp) \) is transformed as followings, where
\[
Gp \equiv G(\text{done}(c_{in}\,?x)) \supset \text{sorted}(x)).
\]
Let \( P_0 \) be as follows.
\[
P_0 \overset{\text{def}}{=} \text{Part}(P, Gp)
\]
By Unfolding and \( G \) rule, the right hand side is transformed as:
\[
\Rightarrow \text{Part}((c_{in}\,?x \rightarrow \text{Part}((c_{out}\,!\text{max}(x) \rightarrow P), Gp)), p)
\]
Applying \( G \) rule again to the underlined part:
\[
\Rightarrow \text{Part}((c_{in}\,?x \rightarrow \text{Part}((c_{out}\,!\text{max}(x) \rightarrow \text{Part}(P, Gp)), p)), p).
\]
By Folding of the underlined part:
\[
\Rightarrow \text{Part}((c_{in}\,?x \rightarrow \text{Part}((c_{out}\,!\text{max}(x) \rightarrow P_0), p)), p).
\]
Applying done rule to the underlined part:
\[
\Rightarrow \text{Part}((c_{in}\,?x \rightarrow \text{Part}((c_{out}\,!\text{max}(x) \rightarrow P_0), p), \text{done}(c_{in}\,?x))) \, p).
\]
By \( \wedge \) rule to the underlined part
\[
\Rightarrow \text{Part}((c_{in}\,?x \rightarrow \text{Part}((c_{out}\,!\text{max}(x) \rightarrow P_0), p \wedge \text{done}(c_{in}\,?x))) \, p)
\]
Applying Termination rule to the whole:
\[
\Rightarrow (c_{in}\,?x \rightarrow \text{Part}((c_{out}\,!\text{max}(x) \rightarrow P_0), p \wedge \text{done}(c_{in}\,?x))).
\]
From \( (p \wedge \text{done}(c_{in}\,?x)) \supset \text{sorted}(x) \), applying \( \supset \) rule to the underlined part
\[
\Rightarrow (c_{in}\,?x \rightarrow \text{Part}((c_{out}\,!\text{max}(x) \rightarrow P_0), \text{sorted}(x))).
\]
As sorted\( (x) \) does not contain any temporal operators, Predicate rule is applied. Assume that the partial evaluation of \( (c_{in}\,?x \rightarrow (c_{out}\,!\text{max}(x) \rightarrow P_0)) \) wrt sorted\( (x) \) by a conventional method replaces max with head. By application of Predicate rule to the underlined part, then we have:
\[
P_0 \overset{\text{def}}{=} (c_{in}\,?x \rightarrow (c_{out}\,!\text{head}(x) \rightarrow P_0)).
\]

4. Soundness

In this section, we prove that the process obtained by the transformation in the previous section behaves similarly to the original process under the constraint.
**Restricted Failure Set Equivalence**

We introduce *restricted failure set equivalence* to formalize the notion that two processes behave equivalently under a given constraint. We also show a number of properties of the equivalence. Restricted failure set equivalence is defined using the notion of failure set equivalence.

**Definition 4.1** [Restriction of a failure set] Let $F$ be a set of pairs $(s, R)$ where $s$ is a sequence of actions and $R$ is a set of actions, and let $q$ be a temporal formula. $F / q$ is a *restriction* of $F$ by $q$ defined as follows.

$$F / q = \{ (s, R) \mid \exists r (s, R) \in F, \ head(r) \in R, \ s^r \models q \}$$

**Definition 4.2** [Restricted Failure Set Equivalence] Let $q$ be a temporal formula. Processes $P_1$ and $P_2$ are *restricted failure set equivalent* w.r.t $q$ if:

$$\text{failures}(P_1) / q = \text{failures}(P_2) / q$$

and denoted $P_1 \sim P_2$ w.r.t $q$.

If $P_1 \sim P_2$ w.r.t $q$ then they behave similarly and no deference can be observed under the environment which satisfies $q$.

**Example 4.1** Let $P$, $P_0$ and $G_p$ be as defined in section 3.2. Example, then $P \sim P_0$ w.r.t $G_p$. The following propositions are easy to prove.

**Proposition 4.1** $\sim$ w.r.t $q$ is an equivalence relation for any $q$.

**Proposition 4.2**

1. Let $P \overset{\text{def}}{=} F$.
   i) If $F \theta \sim Q$ w.r.t $q$ then $P \theta \sim Q$ w.r.t $q$.
   ii) If $F \theta \sim Q$ w.r.t $q$ then $F \theta \sim Q$ w.r.t $q$.

2. Let $P$ be a process that is equal to the process $(m_i \rightarrow P_i)_{i \in I}$.
   Then $P \sim Q$ w.r.t $q$ for every $i \in I$.
3. $P \sim Q$ w.r.t $q$, then $P \mid R \sim Q \mid R$ w.r.t $q$ and $R \mid P \sim R \mid Q$ w.r.t $q$.

These results show that restricted failure set equivalence is a congruence relation for above operations.

**Proposition 4.3**

1. If $P \sim Q$ w.r.t $\text{done}(m)$, then $failures((m \rightarrow P)) = failures((m \rightarrow Q))$.

2. If $P \sim Q$ w.r.t $q_1$ and $Q \sim R$ w.r.t $q_2$, then $P \sim R$ w.r.t $q_1 \land q_2$.

3. For any $P$ and $Q$, if $P \sim Q$ w.r.t $q_1 \land q_2$, then there exists $R$ such that $P \sim R$ w.r.t $q_1$ and $R \sim Q$ w.r.t $q_2$.

4. If $P \sim Q$ w.r.t $q$ and $p \models q$, then $P \sim Q$ w.r.t $p$.

5. $(m_i \rightarrow P_i)_{i \in I} \sim (m_i \theta \rightarrow P_i)$ w.r.t $\text{done}(m_i \theta)$ for $k \in I$ if $m_k \theta \neq m_j \sigma$ for any substitution $\sigma$ and any $j (\in I) \neq k$. 

21
4.2. Soundness of Transformation Rules

We assume that the soundness of the conventional partial evaluation method is already shown. In other words, we assume that if \( P' \) is the result of partial evaluation of \( P \) by (non-temporal) constraint \( q \), then \( P \sim P' \) wrt \( q \).

Proposition 4.4 For any transformation sequence \( E_0, E_1, E_2, \ldots, E_n \), there exists a transformation sequence \( E'_0, E'_1, E'_2, \ldots, E'_{n'} \) such that \( E_0 \) is \( E'_0 \), \( E_n \) is \( E'_{n'} \) and it does not contain any application of \( G \) rule.

Proof: For \( E_0 \), let \( E_{n+1} \) be an expression obtained by replacing a sub expression \( \text{Part}(Q, Gq) \) of \( E_i \) with \( \text{Part}(m_i \to \text{Part}(Q_i, Gq)) \) using \( \text{G rule} \). It is well known that \( Gq = q \land XGq \). Thus, \( \text{Part}(Q, Gq) \) can be replaced with \( \text{Part}(Q, q \land XGq) \) by \( \text{G rule} \). Then \( E_{n+1} \) is obtained using \( \land^- \) rule and \( X \) rule.

Then we can assume that \( G \) \( \text{rule} \) is no longer used in given transformation sequence.

Proposition 4.5 Let \( E_0 \) be \( \text{Part}(P_0, p) \). For a transformation sequence \( E_0, E_1, E_2, \ldots, E_n \), let \( P' \) be a process which is obtained by replacing all sub-expressions \( \text{Part}(Q_1, q_1), \ldots, \text{Part}(Q_k, q_k) \) of the form \( \text{Part}(\ldots) \) in \( E_n \) with \( Q'_1, \ldots, Q'_k \) respectively. If \( Q_j \sim Q'_j \) wrt \( q_i (1 \leq i \leq k) \), then \( P_0 \sim P' \) wrt \( p \).

Proof: We use the induction on \( n \).

1. The case of \( n = 0 \): The only sub expression of the form \( \text{Part}(\ldots) \) is \( E_0 (= \text{Part}(P_0, p)) \) itself. It is replaced with \( P' \) such that \( P_0 \sim P' \) wrt \( p \). The proposition is obvious.

2. The case of \( n > 0 \): \( E_{n+1} \) is an expression that is obtained from \( E_n \) applying one of the rules to the sub-expression \( \text{Part}(Q_i, q_i) \) for some \( i (1 \leq i \leq k) \). Assume that \( P_0 \sim P' \) wrt \( p \) where \( P' \) is obtained from \( E_n \) by replacing \( \text{Part}(Q_i, q_i), \ldots, \text{Part}(Q_k, q_k) \) with \( Q'_1, \ldots, Q'_k \) respectively. Let \( P'' \) be a process obtained from \( E_{n+1} \) by replacing \( \text{Part}(Q_i, q_i), \ldots, \text{Part}(Q_k, q_k) \) with \( Q'_1, \ldots, Q'_k \) respectively. It must be shown that if \( Q_j \sim Q'_j \) wrt \( q_i (1 \leq j \leq k) \), then \( P_0 \sim P'' \) wrt \( p \). We show by cases on the rules that are used on the step from \( E_n \) to \( E_{n+1} \).

The case of Unfolding rule: From \( P \theta \Rightarrow F \theta \), \( \text{Part}(P \theta, q_i) \) is replaced with \( \text{Part}(F \theta, q_i) \) in \( E_{n+1} \). Now \( \text{Part}(F \theta, q_i) \) is replaced by \( F' \) such that \( F \theta \sim F' \) wrt \( q_i \), then we obtain \( P'' \). From Proposition 4.2, 1. i), \( P \theta \sim F' \) wrt \( q_i \). Then, \( P'' \) is identical with \( P' \) that can be obtained from \( E_n \) by replacing \( \text{Part}(P \theta, q_i) \) with \( F' \). From the inductive hypothesis, \( P \sim P' \) wrt \( p \). Then, \( P \sim P'' \) wrt \( p \).

The case of Folding rule: Similar to the above case using Proposition 4.2, 1. ii).

The case of Predicate rule: \( \text{Part}(Q_i, q_i) \) is replaced with \( Q_{qi} \) in \( E_{n+1} \), where \( Q_{qi} \) is the partial evaluation of \( Q_i \) by \( q_i \) (with the conventional method). From the assumption given at beginning of this section, \( Q_i \sim Q_{qi} \) wrt \( q_i \). Thus \( P'' \) is identical with \( P' \) that is obtained from \( E_n \) by replacing \( \text{Part}(Q_i, q_i) \) with \( Q_{qi} \) which is \( Q_i \sim Q_{qi} \) wrt \( q_i \). From the inductive hypothesis, \( P \sim P' \) wrt \( p \).

The case of done rule: \( E_{n+1} \) contains all sub expressions \( \text{Part}(\ldots) \) that appeared in \( E_n \). In addition to them, \( E_{n+1} \) also has sub expressions \( (m \to \text{Part}(P, \text{done}(m))) \) which are introduced by replacing sub expressions \( (m \to P) \) in \( E_n \). Let \( P'' \) be an expression that is obtained by replacing \( \text{Part}(Q_1, q_1), \ldots, \text{Part}(Q_k, q_k) \) in \( E_n \) with \( Q'_j (\sim Q_j \text{ wrt } q_i (1 \leq j \leq k)) \), respectively and replacing \( \text{Part}(P, \text{done}(m)) \) with \( Q'(\sim P \text{ wrt } \text{done}(m)) \) from Proposition 4.3, 1. i). Then \( \text{failures}(m \to P) = \text{failures}(m \to Q) \). Thus for \( P' \) and \( P'' \), \( \text{failures}(P') = \text{failures}(P'') \). By inductive hypothesis, \( P \sim P'' \) wrt \( p \).
The case of \( \land^- \) rule: In this case \( \text{Part}(Q, q^1 \land q^2) \) is replaced with \( \text{Part}(\text{Part}(Q, q^1), q^2) \) in \( E_{m+1} \). \( P'' \) is obtained by replacing \( \text{Part}(Q, q^1) \) with \( Q' \); such that \( Q' \sim Q \) wrt \( q^1 \) at first, then \( \text{Part}(Q', q^2) \) is replaced with \( Q'' \); such that \( Q'' \sim Q' \) wrt \( q^2 \). From Proposition 4.3, 2., \( Q'' \sim Q \), \( q^1 \land q^2 \). Then it is the case from the inductive hypothesis as above case.

The case of \( \land^+ \) rule: In this case, \( \text{Part}(\text{Part}(P, q^1), q^2) \) in \( E_n \) is replaced with \( \text{Part}(P, q^1 \land q^2) \) in \( E_{n+1} \). Let \( P'' \) be an expression that is obtained by the same manner to the case of \( E_n \) to \( P' \) but replacing \( P \) with \( Q \), such that \( P \sim Q \) wrt \( q^1 \land q^2 \). From Proposition 4.3, 3., if \( P \sim Q \) wrt \( q^1 \land q^2 \) then there exists \( R \) such that \( P \sim R \) wrt \( q^1 \) and \( R \sim Q \) wrt \( q^2 \). Thus, \( P'' \) is obtained from \( E_n \) by replacing \( \text{Part}(P, q^1) \) with \( R \) and replacing \( \text{Part}(R, q^2) \) with \( Q \) again. By inductive hypothesis, the proposition holds.

The case of \( \Rightarrow \) rule: In this case \( \text{Part}(P, p) \) in \( E_n \) is replaced with \( \text{Part}(P, q) \) in \( E_{n+1} \). Let \( P'' \) be an expression that is obtained by the same manner to the case of \( E_n \) to \( P' \) but replacing \( P \) with \( Q \), such that \( P \sim Q \) wrt \( q \) from Proposition 4.3, 4., if \( P \sim Q \) wrt \( q \) and \( p \Rightarrow q \), then \( P \sim Q \) wrt \( p \). Thus \( P'' \) is obtained by replacing \( \text{Part}(P, p) \) with \( Q_i \) in \( E_n \). From the inductive hypothesis, the proposition holds.

The case of \( \times \) rule: \( \text{Part}(Q, Xq) \) in \( E_n \) is replaced with \( (m_i \rightarrow \text{Part}(Q_i, q))_{j \in J} \) in \( E_{n+1} \) where \( Q \) is a process that is equal to \( (m_i \rightarrow Q_i)_{j \in J} \). \( P'' \) is obtained from \( E_{n+1} \) by replacing each \( \text{Part}(Q, q) \) with \( Q'_i \) such that \( Q'_i \sim Q_i \) wrt \( q \) respectively for each \( i \in J \). From Proposition 4.2, 2., \( (m_i \rightarrow Q'_i)_{j \in J} \sim Q \) wrt \( Xq \). Then \( P'' \sim P_0 \) wrt from the inductive hypothesis.

The case of Pruning rule: \( \text{Part}(Q, q_i) \) in \( E_n \) is replaced with \( (m_i \theta \rightarrow P_i) \) in \( E_{n+1} \) where \( Q_i \) is \( (m_i \rightarrow P_i)_{j \in J} \) and \( q_i \) is Xdone(\( m_i \theta \)) for \( i \in J \). From Proposition 4.3, 5., \( (m_i \rightarrow P_i)_{j \in J} \sim (m_0 \rightarrow P_i) \) wrt Xdone(\( m_0 \theta \)). Then \( P'' \sim P_0 \) wrt from the inductive hypothesis as \( P'' \) is obtained similarly to \( P' \) but replacing \( \text{Part}(Q, q_i) \) by \( (m_i \theta \rightarrow P_i) \) in \( E_n \).

The case of \( \sqcup \) rule: For \( \text{Part}(Q, q_i) \), \( Q \) is \( Q^1 \sqcup Q^2 \). \( \text{Part}(Q, q_i) \) is replaced with \( \text{Part}(Q^1, q_i) \sqcup \text{Part}(Q^2, q_i) \). \( P'' \) is obtained from \( Q^1 \sqcup Q^2 \) with \( Q^1 \sim Q^1 \) wrt \( q_i \) and \( Q^2 \sim Q^2 \) wrt \( q_i \) in \( P'' \). From Proposition 4.2, 3., \( Q^1 \sqcup Q^2 \sim Q^2 \sqcup Q^2 \) wrt \( q_i \). Then, \( P'' \sim P_0 \) wrt from the inductive hypothesis.

The case of Termination rule: In this case, for any \( Q_i \) and \( q_i \), \( \text{Part}(Q, q_i) \) is replaced with \( Q \) itself. As \( \rightsquigarrow \) is reflexive for all \( q_i \), \( P'' \) is obtained from \( E_n \) by replacing \( \text{Part}(Q, q_i) \) with \( Q_i \) that is equivalent to itself wrt \( q_i \) and similar to \( P' \) for other parts. Then, \( P'' \sim P \) wrt \( q_i \) from the inductive hypothesis.

Theorem 4.1 [Soundness] For any process \( P \) and the constraint \( p \), if \( Q \) is obtained from \( \text{Part}(P, p) \) by applying the rules, then \( P \sim Q \) wrt \( p \).

Proof: \( Q \) is \( E_n \) of Proposition 4.5 without sub expressions of the form \( \text{Part}(\ldots) \).

From Theorem 4.1, we have that \( P \) and \( P_0 \) in section 3.2 Example is \( P \sim P_0 \) wrt \( \forall p \).

5. APPLICATION FOR IMPROVEMENT OF SECURITY

This section presents an example of an application of our partial evaluation method to improve the security of a concurrent system. It is an example of authentication protocol that is a modification of Needham-Schroeder-Lowe protocol[8,14].

Example 5.1. Consider an example consists of three agents Alice, Bob and Charlie. Alice and Bob establish a connection with authentication using public key cryptosystem, and let Charlie be
Every communication between Alice and Bob is transferred via Charlie. First, we consider the case that Charlie is not malicious and he just forwards massages from Alice to Bob or from Bob to Alice. In this case, Charlie is regarded as a part of the trusted network.

The protocol is as follows (Figure 3). First, Alice sends a request \( R \) to get Bob’s public key \( K_B \). Bob sends \( K_B \) as the reply to \( R \). Alice sends the message \([Id_A, N_A]_{KB}\) encrypted with \( K_B \) where \( Id_A \) is her own id, \( N_A \) is a nonce. Bob receives \([Id_A, N_A]_{KB}\) and decrypts the message. Then he gets \( N_A \). He sends the message \([Id_B, N_A, N_B]_{KA}\) consists of his own id \( Id_B \), \( N_A \) and a new nonce \( N_B \) encrypting with Alice’s public key \( K_A \) (We assume Bob already has Alice’s public key for the simplicity). Alice receives the message and she gets \( N_B \). She sends \( N_B \) encrypting with \( K_B \) as \([N_B]_{KB}\). Then Bob receives \([N_B]_{KB}\) and then the authentication is completed.

The system consists of Alice and Bob is defined as the process \( \text{AliceBob} \) that communicates with the process of Charlie. Let \( c_A \) be the channel name for communication from Alice to Charlie, and \( a \) be the channel from Charlie to Alice. \( c_B \) and \( b \) are channels for the communication of Charlie and Bob. In the followings, terms begin with capital letters such as \( N_A, N_B, Id_A, Id_B, K_A, K_B \) and \( K_C \) denotes values of nonces, id’s or public keys respectively. Terms begin with lower case letters such as \( id_A, id_B, n_A, n_B, k, k_B, \ldots \) are variables to receive id’s, nonces or keys respectively.

**AliceBob** \( \overset{\text{def}}{=} (c_A!R \rightarrow (b?r \rightarrow (c_B!K_B \rightarrow (a?k \rightarrow \\
(c_A![Id_A, N_A]_{KB} \rightarrow (b?[id_A, n_A]_{KB} \rightarrow \\
(c_B![Id_B, n_A, N_B]_K_A \rightarrow (a?[id_B, N_A, n_B]_{KA} \rightarrow \\
(c_A![n_B]_{KB} \rightarrow (b?[N_B]_{KB} \rightarrow \text{OK}))))))))))) \)

In this case, Alice is so incautious that she ignores the value of \( id_B \) and does not check the validity of public key \( k \) just as the original Needham-Schroeder protocol[19] with the security hole which [14] reported.

**The man in the middle** Charlie is the process as follow if he is not malicious.

\( \text{C} \overset{\text{def}}{=} (c_A?r \rightarrow (b!r \rightarrow (c_B?k_B \rightarrow (a!k_B \rightarrow (c_A?m_A \rightarrow (b!m_A \rightarrow \\
(c_B!m_B \rightarrow (a!m_B \rightarrow (c_A?m^2_A \rightarrow (b!m^2_A \rightarrow C))))))))))) \)

where \( r \) is a variable for the request, \( m_A^j \ (j = 1, 2) \) are variables for the messages from Alice and \( m_B \) is a variable for the message from Bob. He just forwards the messages for each direction.
On the other hand, we consider the case that Charlie is malicious. The following process \( C' \) is the behaviour of malicious Charlie making the “man-in-the-middle attack”.

\[
C' \overset{\text{def}}{=} C \parallel C''
\]

where

\[
C'' \overset{\text{def}}{=} (c_a \diamond r \rightarrow (b! r \rightarrow (c_b ? k_B \rightarrow (a! K_C \rightarrow (c_a ? m_1^A \rightarrow (b! [a, n_A]_K_B \rightarrow (c_a ? m_B \rightarrow (a! m_B \rightarrow (c_a ? m_2^A \rightarrow (b! [n_B]_K_B \rightarrow \text{fakeBob})))))))))).
\]

![Figure 4. A Man in The Middle Attack](image)

As \( C'' \) is a nondeterministic process, sometimes he may act as \( C \), but he may act maliciously. In the case of malicious Charlie, \( C' \) replies his own public key \( K_C \) instead of Bob's key for Alice’s request. As the Alice’s message \( m_1^A \) is \([Id_A, N_A]_{K_C}\) that is encrypted with Charlie’s key \( K_C \), he can decrypt it and get \( A \) and \( N_A \). He send Bob them as \([a, n_A]_{K_B}\). After forwarding Bob’s reply to Alice, he get \( N_B \) from \( m_2^A \) as it is encrypted with \( K_C \) again. So he get both of \( N_A \) and \( N_B \), he can pretend to be Bob (Figure 4).

If Alice is cautious, the system of Alice and Bob is defined as follows.

\[
\text{AliceBob'} \overset{\text{def}}{=} (c_a \diamond R \rightarrow (b? r \rightarrow (c_b ! K_B \rightarrow (a? k \rightarrow (c_a ![Id_A, N_A]_K_a \rightarrow (b? [Id_A, n_A]_{K_a} \rightarrow ((a? [Id_B, n_B]_K_B \rightarrow (b? [Id_B, N_B]_{K_B} \rightarrow \text{OK}))))))).
\]

She checks Bob’s message if the id is same to the owner of the received key \( k \). If Charlie is as \( C \) (not malicious), \( k \) is Bob’s key and the id of the owner of \( k \) is equal to \( Id_B \) that she receives. If the owner of \( k \) is not equal to the \( Id_B \), the she refuses to receive the message and does not proceed the protocol.

For these definitions, when \( \text{AliceBob'} \) behaves satisfying \( \text{goodC} \) such that:

\[
\text{goodC} \equiv \chi (\text{done}(c_a ! R) \land \chi (\text{done}(b? R) \land \chi (\text{done}(c_b ! K_B) \land \chi (\text{done}(a? K_a)) \land \chi (\text{done}(c_a ![Id_A, N_A]_K_a) \land (b? [Id_A, n_A]_{K_a}) \land ((a? [Id_B, n_B]_K_B) \rightarrow (b? [Id_B, N_B]_{K_B} \rightarrow \text{OK})))))
\]
C' acts as C because Alice receives K_B at a?k that is same to the value sent by c_B! K_B and the id of owner of this key is Id_B that she receives. Then Alice and Bob establish the connection and we have

$$ AliceBob \sim AliceBob' \text{ wrt goodC.} $$

Namely, they are equivalent if Charlie is not malicious. On the other hand, when C' does a!K_C after c_B?k K_B, AliceBob'||C' behaves differently from AliceBob||C'. It deadlocks without establishing the insecure connection.

Thus, we consider that for a process P, the secure version of P is obtained as the process P' such that P' detects attacks and P' ~ P wrt q where q is the constraint that there is no malicious agent who communicates with P or P'. Our partial evaluation method gives a process Q such that Q ~ P wrt q as presented in the previous section. So we consider that the partial evaluation method is useful to improve not only the efficiency of process but also to improve the security of system. In the case of this example, both of Part((AliceBob'), goodC) and Part((AliceBob, goodC) can be transformed into the following process AliceBob'' using X rule and Pruning rule etc. presented in section 3.1. Note that in this process, Alice knows K_B and Id_B before she receive them.

$$ AliceBob'' = \text{def} \left( c_A!R \rightarrow (b?r \rightarrow (c_B!K_B \rightarrow (a? K_B \rightarrow \right.$$ 

$$ (c_A![Id_A, N_A]_{K_B} \rightarrow (b?[id_A, n_A]_{K_B} \rightarrow \right.$$ 

$$ (c_B![Id_B, n_B, N_B]_{K_A} \rightarrow (a?[Id_B, N_A, n_B]_{K_A} \rightarrow \right.$$ 

$$ (c_A![n_B]_{K_B} \rightarrow (b?[N_B]_{K_B} \rightarrow \text{OK}))))))) \right)$$

From Theorem 4.1, we have

$$ AliceBob'' \sim AliceBob \text{ wrt goodC and AliceBob'' \sim AliceBob'} \text{ wrt goodC.} $$

Then we have

$$ AliceBob \sim AliceBob' \text{ wrt goodC} $$

from Proposition 4.1.

6. CONCLUSIONS

A set of the rules for partial evaluation of CSP processes using temporal formulas and its soundness are presented. We can specialize reactive processes using constrains on the input messages which are delivered in the middle of execution. The set of rules presented here can be used with any conventional partial evaluation method for CSP like languages if it preserves restricted failure set equivalence. Thus the set of rules of this paper can be regard as a framework to extend partial evaluation methods for transformational programs to reactive processes. We also mentioned the relation to the improvement of security.

REFERENCES


Zobel D., Program Transformations for Distributed Control Systems, citeceerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.2473

Authors

Masaki Murakami is an associate professor of Division of Logic and Computation in Department of Information Technology of Okayama University since 1991. His current research interests are the theory of concurrency, formal semantics of programming languages and program transformation of concurrent programs. He graduated Department of Information Engineering of Nagoya Institute of Technology in 1980. He received his Dr. of Engineering from Nagoya University. He joined Fifth Generation Computer Project from 1985 to 1989. He was working for Fujitsu from 1989 to 1991.