Undecidability of 2-Label BPP Equivalences and Behavioral Type Systems for the π-Calculus

Naoki Kobayashi and Takashi Suto

Tohoku University
{koba,tsuto}@kb.ecei.tohoku.ac.jp

Abstract. The trace equivalence of BPP was shown to be undecidable by Hirshfeld. We show that the trace equivalence remains undecidable even if the number of labels is restricted to two. The undecidability result holds also for the simulation of two-label BPP processes. These results imply undecidability of some behavioral type systems for the π-calculus.

1 Introduction

BPP [2] is a process calculus which has prefixes (lP), sum, parallel composition, and recursion as process constructors. Hirshfeld [3] has shown that the trace equivalence of two BPP processes is undecidable, by encoding the halting problem of a Minsky machine into the trace inclusion relation between two BPP processes. Hüttel [4] extended the undecidability result to other preorders between processes.

In this paper, we show that the trace inclusion of BPP processes remains undecidable even if we restrict the number of action labels to two. In the rest of the paper, we call the restriction of BPP to two labels *2-label BPP*. Hirshfeld's encoding of a Minsky machine requires 6 action labels, hence his result does not immediately extend to the case of 2-label BPP processes.

One may think that the undecidability for 2-label BPP processes can be easily obtained by encoding an action label into a sequence of the two labels, so that $P \leq_{tr} Q$ if and only if $[\![P]\!] \leq_{tr} [\![Q]\!]$, where $P \leq_{tr} Q$ means that the trace set of P is a subset of the trace set of Q, and $[\![P]\!]$ is the 2-label BPP process obtained from P by using the label encoding. Then, the undecidability of the trace inclusion for 2-label BPP (and hence also the undecidability of the trace equivalence) would follow from the undecidability for general BPP processes. We basically follow this approach, but there are two main difficulties. First, because of the existence of parallel composition, encoding of some action of a process may be simulated by interleaving execution of encodings of two or more actions of the other process. For example, consider two processes $P_1 = l_2 \,|\, l_2$ and $Q_1 = (l_2 \,|\, l_2) + l_3 l_1 l_1$ and choose the following label encoding: $[\![l_1]\!] = a, [\![l_2]\!] = ba, [\![l_3]\!] = bb$. Then, the trace sets of P_1 and Q_1 are of course different, but the trace sets of $[\![P_1]\!] = ba \,|\, ba$ and $[\![Q_1]\!] = (ba \,|\, ba) + bbaa$ are equivalent. Second, a naive encoding may also invalidate the equivalence of processes. For example, consider $P_2 = l_2 \,|\, l_2$ and $Q_2 = l_2 l_2$. These have the same trace sets (and they are even bisimilar). With the above encoding, however, $[\![P_2]\!]$ has the trace *bbaa* while $[\![Q_2]\!]$ does not. To overcome the first problem, we choose an encoding of labels such that a shuffle of two or more encoded labels (i.e., a partial trace of $[\![l_1]\!] \,|\, \cdots \,|\, [\![l_m]\!]$) cannot be confused with encoding $[\![l]\!]$ of a single action. To avoid the second problem, we prepare a process *Inv* that simulates invalid sequences. With *Inv*, we can establish that $P \leq_{tr} Q$ if and only if $[\![P]\!] \leq_{tr} [\![Q]\!] \,|\, \mathit{Inv}$, since *Inv* simulates all the invalid sequences of $[\![P]\!]$ (which are generated by interleaving execution of more than one encoded actions). A similar (but a little more complicated) technique can also be used to show the undecidability of the simulation preorder of 2-label BPP processes.[1]

As an application of the undecidability results above, we show that the type checking problems for some behavioral type systems for the π-calculus are also undecidable.[2] In the behavioral type systems, channel types are augmented with *usage expressions* (usages, in short), describing how each communication channel is used. The usages can be regarded as 2-label BPP processes. Since the trace preorder between two usages can be reduced to the typability of a certain process, the type checking problem is undecidable.

The rest of this paper is structured as follows. Section 2 introduces BPP. Section 3 proves the undecidability of trace inclusion of 2-label BPP. Section 4 applies a similar technique to prove that the simulation preorder is also undecidable for 2-label BPP. Section 5 applies the undecidability results to show that certain behavioral type systems for the π-calculus are undecidable. Section 6 discusses related work, and Section 7 concludes. Full proofs are found in the full version of this paper [11].

2 Basic Parallel Processes (BPP)

BPP [2] is a calculus of processes consisting of prefixes, parallel composition, internal choice, and recursion. Unlike in CCS [13], there is no synchronization mechanism (such as the transition $a.P \,|\, \overline{a}.Q \xrightarrow{\tau} P \,|\, Q$).

The syntax of processes is given by:

$$P ::= \mathbf{0} \mid X \mid lP \mid (P|Q) \mid P + Q \mid \mu X.P$$

Here, X and l are meta-variables ranging over the sets of *process variables* and *action labels* respectively. We write **Act** for the set of action labels, and write $\mathbf{BPP_{Act}}$ for the set of BPP processes whose action labels are in the set **Act**.

The process $\mathbf{0}$ does nothing. A process lP first performs l and then behaves like P. $P|Q$ is a parallel composition of P and Q, and $P + Q$ is an internal choice of P or Q. $\mu X.P$ stands for the recursive process X usually defined by the equation $X = P$.

[1] Note that the trace inclusion (as well as the simulation preorder) is decidable for 1-label BPP processes.

[2] Actually, investigation into the type checking problems lead us to the study of the trace and simulation preorders for 2-label BPP in this paper.

$$\frac{}{lP \xrightarrow{l} P} \quad (\textsc{Tr-Act}) \qquad \frac{[\mu X.P/X]P \xrightarrow{l} Q}{\mu X.P \xrightarrow{l} Q} \quad (\textsc{Tr-Rec})$$

$$\frac{P \xrightarrow{l} P'}{P \,|\, Q \xrightarrow{l} P' \,|\, Q} \quad (\textsc{Tr-ParL}) \qquad \frac{Q \xrightarrow{l} Q'}{P \,|\, Q \xrightarrow{l} P \,|\, Q'} \quad (\textsc{Tr-ParR})$$

$$\frac{P \xrightarrow{l} P'}{P + Q \xrightarrow{l} P'} \quad (\textsc{Tr-OrL}) \qquad \frac{Q \xrightarrow{l} Q'}{P + Q \xrightarrow{l} Q'} \quad (\textsc{Tr-OrR})$$

Fig. 1. Transition rules of BPP processes

We often omit $\mathbf{0}$ and just write a for $a\mathbf{0}$. We give a higher-precedence to unary prefixes, $+$, and $|$ in this order, so that $l_1P_1|l_2P_2 + l_3P_3$ means $(l_1P_1)|((l_2P_2) + (l_3P_3))$.

We say that P is guarded by l in lP. A recursive process $\mu X.P$ is *guarded* if X appears only in guarded positions of P. A process is *guarded* if all its recursive processes are guarded. In the rest of this paper, we consider only closed, guarded processes.[3]

The transition relation $P \xrightarrow{l} Q$ is the least relation closed under the rules in Figure 1. We write $P \xrightarrow{l_1 \cdots l_n} Q$ if $P \xrightarrow{l_1} \cdots \xrightarrow{l_n} Q$.

2-label BPP is BPP where the set **Act** of action labels is restricted to the set $\{a, b\}$. Hence, the set of 2-label BPP processes is $\mathbf{BPP}_{\{a,b\}}$.

3 Undecidability of Trace Equivalence

In this section, we show that the trace equivalence of 2-label BPP processes is undecidable. As sketched in Section 1, we show an encoding of general BPP processes into 2-label BPP processes, so that the trace preorder is preserved. Then, the undecidability follows from the undecidability result for general BPP [3]. The undecidability of the trace equivalence can be shown also by using the encoding in Section 4, but the encoding presented in this section is simpler and easier to understand.

3.1 Trace Set, Trace Preorder, and Trace Equivalence

Definition 1 (trace set) The *trace set* of P, written $\mathit{traces}(P)$, is defined by:

$$\mathit{traces}(P) = \{l_1 \ldots l_n \mid P \xrightarrow{l_1} \cdots \xrightarrow{l_n} P_n\}$$

[3] Actually, any recursive process can be transformed to a bisimilar, guarded recursive process. For example, $\mu X.(X \,|\, lX)$ is equivalent to the guarded process $\mu X.l(X \,|\, X)$. $\mu X.X$ is bisimilar to $\mathbf{0}$.

Definition 2 The *trace preorder* $\leq_{tr}$ and the *trace equivalence* $\sim_{tr}$ are defined by:

$$P \leq_{tr} Q \overset{def}{\Leftrightarrow} traces(P) \subseteq traces(Q)$$

$$P \sim_{tr} Q \overset{def}{\Leftrightarrow} P \leq_{tr} Q \land Q \leq_{tr} P$$

3.2 Encoding

We first define the encoding of labels. Since the number of labels occurring in a given process is finite, we assume here that the set **Act** of action labels is a finite set $\{l_0, \dots, l_{N-1}\}$. In the rest of this section and Section 4, we use meta-variables $P, Q, \dots$ for processes in $\mathbf{BPP}_{\{l_0,\dots,l_{N-1}\}}$,and use meta-variables $E, F, \dots$ for processes in $\mathbf{BPP}_{\{a,b\}}$.

Definition 3 *A mapping* $[\![\cdot]\!]$ *from* **Act** *to* $\{a, b\}^*$ *is defined by:*

$$[\![l_i]\!] = ab^i ab^{2N-1-i}$$

Here, a^i stands for the sequence of a of length i. For example, $a^3 = aaa$.

We now define encoding of a process. As mentioned in Section 1, we use different encodings for P and Q in $P \leq_{tr} Q$.

Definition 4
Mappings $[\![\cdot]\!]_L$ *and* $[\![\cdot]\!]_R$ *from* $\mathbf{BPP}_{\{l_0,\dots,l_{N-1}\}}$ *to* $\mathbf{BPP}_{\{a,b\}}$ *are defined by:*

$$\begin{array}{ll}
[\![\mathbf{0}]\!]_L = \mathbf{0} & [\![P \,|\, Q]\!]_L = [\![P]\!]_L \,|\, [\![Q]\!]_L \\
[\![X]\!]_L = X & [\![P + Q]\!]_L = [\![P]\!]_L + [\![Q]\!]_L \\
[\![lP]\!]_L = [\![l]\!][\![P]\!]_L & [\![\mu X.P]\!]_L = \mu X.[\![P]\!]_L \\
[\![P]\!]_R = [\![P]\!]_L \,|\, Inv & \\
\multicolumn{2}{l}{\textit{where } Inv = \sum_{k<N, k+l<2N-1} ab^k ab^l aG \textit{ and } G = \mu X.(aX + bX)}
\end{array}$$

The role of the process Inv in $[\![P]\!]_R$ is to simulate invalid transition sequences (caused by interleaving execution of $[\![l_i]\!]$ and $[\![l_j]\!]$).

3.3 Undecidability of Trace Equivalence

The main result of this section is stated as follows.

Theorem 1. *$P \leq_{tr} Q$ if and only if $[\![P]\!]_L \leq_{tr} [\![Q]\!]_R$.*

Since $P \leq_{tr} Q$ is undecidable for general BPP [3], we obtain the following corollary.

Corollary 1. *The trace inclusion $E \leq_{tr} F$ and trace equivalence $E \sim_{tr} F$ are undecidable for 2-label BPP.*

Proof. If the trace inclusion $\leq_{tr}$ were decidable for 2-label BPP, then we could decide $P \leq_{tr} Q$ for general BPP by deciding $[\![P]\!]_L \leq_{tr} [\![Q]\!]_R$, hence a contradiction. To see that $E \sim_{tr} F$ is also undecidable, it suffices to observe that $E \leq_{tr} F$ if and only if $E + F \sim_{tr} F$. □

The rest of this section is devoted to the proof of Theorem 1. The followings are key lemmas needed to prove Theorem 1.

Lemma 1. *Let $m \in \{L, R\}$. If $P \xrightarrow{l} Q$, then $[\![P]\!]_m \xrightarrow{[\![l]\!]} [\![Q]\!]_m$.*

Lemma 2. *Let $m \in \{L, R\}$. If $[\![P]\!]_m \xrightarrow{[\![l]\!]} E$, then there exists a process Q such that $E = [\![Q]\!]_m$ and $P \xrightarrow{l} Q$.*

Lemma 1, which follows by straightforward induction on the derivation of $P \xrightarrow{l} Q$, says that any transition of P can be simulated by $[\![P]\!]_L$ and $[\![P]\!]_R$. Lemma 2 says that any *valid* (in the sense that the transition label sequence corresponds to a label of P) transition sequence of $[\![P]\!]_L$ or $[\![P]\!]_R$ can be simulated by P.

Lemma 2 follows by induction on the derivation of the first transition of $[\![P]\!]_m \xrightarrow{[\![l]\!]} E$; See [11] for the full proof of Lemma 2. The proof makes use of the following key property, which essentially says that the first problem mentioned in Section 1 (that a single action may be simulated by interleaving execution of two or more actions) cannot occur.

Lemma 3. *If $[\![P_1 \,|\, P_2]\!]_L \xrightarrow{[\![l]\!]} E$, then either (i) $[\![P_1]\!]_L \xrightarrow{[\![l]\!]} E_1$ and $E = E_1 \,|\, [\![P_2]\!]_L$ or (ii) $[\![P_2]\!]_L \xrightarrow{[\![l]\!]} E_2$ and $E = [\![P_1]\!]_L \,|\, E_2$*

Proof sketch By the transition rules, we have: (i) $[\![P_1]\!]_L \xrightarrow{s_1} E_1$, (ii) $[\![P_2]\!]_L \xrightarrow{s_2} E_2$, (iii) $E = E_1 \,|\, E_2$, and (iv) $[\![l]\!]$ is a shuffle of s_1 and s_2. It suffices to show that either s_1 or s_2 is an empty sequence. Suppose that s_1 and s_2 are not empty. Then s_1 and s_2 must be of the form ab^{j_1} and ab^{j_2} where $j_1, j_2 \leq N - 1$. Then, $[\![l]\!]_L$ cannot be a shuffle of s_1 and s_2, since $[\![l]\!]_L$ contains $2N - 1$ occurrences of b, whereas $j_1 + j_2 \leq 2N - 2$. □

We state another key lemma below. Let $\mathbf{InvTr} = \{s \in \{a, b\}^* \mid \neg\exists s', l.(s = [\![l]\!]s')\}$. In other words, $\mathbf{InvTr}$ is the set of label sequences whose prefixes do not match $[\![l]\!]$.

Lemma 4. *If $s \in \mathbf{InvTr} \cap traces([\![P]\!]_L)$, then $s \in traces(Inv)$.*

Lemma 4 means that any initially invalid sequence generated by $[\![P]\!]_L$ can be simulated by *Inv*. Thus, the second problem mentioned in Section 1 is resolved.

We can now prove Theorem 1.

Proof of Theorem 1

– "Only if": Suppose $P \leq_{tr} Q$ and $s \in traces([\![P]\!]_L)$. We need to show $s \in traces([\![Q]\!]_R)$. s must be of the form $[\![l_{k_1}]\!] \cdots [\![l_{k_n}]\!] s'$ where $s' \in \mathbf{InvTr}$.

By Lemma 2, there exists P_1 such that $P \xrightarrow{l_{k_1}\cdots l_{k_n}} P_1$ and $s' \in \mathit{traces}(\llbracket P_1 \rrbracket_L)$. By the assumption, there must exist Q_1 such that $Q \xrightarrow{l_{k_1}\cdots l_{k_n}} Q_1$. By using Lemma 1, we get $\llbracket Q \rrbracket_R \xrightarrow{\llbracket l_{k_1}\cdots l_{k_n} \rrbracket} \llbracket Q_1 \rrbracket_R$. By Lemma 4, we have $s' \in \mathit{traces}(\mathit{Inv}) \subseteq \mathit{traces}(\llbracket Q_1 \rrbracket_R)$. Thus, we have $s \in \mathit{traces}(\llbracket Q \rrbracket_R)$ as required.

– "If": Suppose $\llbracket P \rrbracket_L \leq_{tr} \llbracket Q \rrbracket_R$ and $l_{k_1} \cdots l_{k_n} \in \mathit{traces}(P)$. By Lemma 1 $\llbracket l_{k_1} \rrbracket \cdots \llbracket l_{k_n} \rrbracket \in \mathit{traces}(\llbracket P \rrbracket_L)$. By the assumption $\llbracket P \rrbracket_L \leq_{tr} \llbracket Q \rrbracket_R$, we have $\llbracket l_{k_1} \rrbracket \cdots \llbracket l_{k_n} \rrbracket \in \mathit{traces}(\llbracket Q \rrbracket_R)$. By using Lemma 2, we obtain $l_{k_1} \cdots l_{k_n} \in \mathit{traces}(Q)$ as required.

□

4 Undecidability of Simulation Equivalence

In this section, we show that the simulation preorder and equivalence are also undecidable for 2-label BPP. We use the undecidability of the simulation preorder for the general BPP [4].[4]

Definition 5 *A binary relation $\mathcal{R}$ on BPP processes is a* simulation *if, for any P, Q, l such that $P \mathcal{R} Q$ and $P \xrightarrow{l} P'$, there exists Q' such that $Q \xrightarrow{l} Q'$ and $P' \mathcal{R} Q'$. The* simulation preorder *$\leq_{sim}$ is the union of all simulations, i.e., $P \leq_{sim} Q$ if and only if there exists a simulation $\mathcal{R}$ such $P \mathcal{R} Q$. We write $P \sim_{sim} Q$ if $P \leq_{sim} Q \wedge Q \leq_{sim} P$.*

Note that $\leq_{sim}$ itself is a simulation (hence, the largest simulation).

We show the undecidability of the simulation preorder for 2-label BPP, by reduction of the simulation preorder for general BPP into that for 2-label BPP. We first need to change the encoding $\llbracket \cdot \rrbracket_R$ of the right-hand side process.

Definition 6 *A mapping $\llbracket \cdot \rrbracket_{R'}$ from $\mathbf{BPP}_{\{l_0,\ldots,l_{N-1}\}}$ to $\mathbf{BPP}_{\{a,b\}}$ is defined by:*

$$\begin{array}{rlrl}
\llbracket \mathbf{0} \rrbracket_{R'} &= \mathbf{0} & \llbracket P \rrbracket^{\epsilon,k_1,k_2} &= \llbracket P \rrbracket_{R'} \\
\llbracket X \rrbracket_{R'} &= X & \llbracket P \rrbracket^{as,1,k} &= a\llbracket P \rrbracket^{s,2,k} + aH^{(2N-2-k)} \\
\llbracket lP \rrbracket_{R'} &= a\llbracket P \rrbracket^{s,1,0} \quad (as = \llbracket l \rrbracket) & \llbracket P \rrbracket^{bs,1,k} &= b\llbracket P \rrbracket^{s,1,k+1} + aH^{(2N-2-k)} \\
\llbracket P \,|\, Q \rrbracket_{R'} &= \llbracket P \rrbracket_{R'} \,|\, \llbracket Q \rrbracket_{R'} & \llbracket P \rrbracket^{bs,2,k} &= b\llbracket P \rrbracket^{s,2,k+1} + aG \\
\llbracket P + Q \rrbracket_{R'} &= \llbracket P \rrbracket_{R'} + \llbracket Q \rrbracket_{R'} & H^{(k)} &= \begin{cases} aG \quad (k=0) \\ bH^{(k-1)} + aG \quad (k > 0) \end{cases} \\
\llbracket \mu X.P \rrbracket_{R'} &= \mu X.\llbracket P \rrbracket_{R'} & &
\end{array}$$

Note that the process G has been defined in Definition 4.

Intuitively, $\llbracket P \rrbracket^{s,k_1,k_2}$ represents an intermediate state for simulating a single action of the original process. The sequence $s \in \{a,b\}^*$ is the remaining sequence of actions to be performed, and k_1 and k_2 are the numbers of a and b actions that have been already performed. The roles of $aH^{(2N-2-k)}$ and aG in the definitions of $\llbracket P \rrbracket^{s,k_1,k_2}$ are to simulate invalid transitions.

[4] The proofs in [4] contain some flaws, but the undecidability results are valid. Please refer to [12] for the flaws and corrected proofs of the undecidability for the general BPP.

Theorem 2. *$P \leq_{sim} Q$ if and only if $[\![P]\!]_L \leq_{sim} [\![Q]\!]_{R'}$.*

To show the "if" part, it suffices to show that the relation $\{(P,Q) \mid [\![P]\!]_L \leq_{sim} [\![Q]\!]_{R'}\}$ is a simulation. To show the "only if" part, we use the following, standard up-to technique:

Lemma 5 (up-to technique). *Let $\mathcal{R}$ be a binary relation on BPP processes. If $\mathcal{R}$ satisfies:*

$$\forall P, Q, P', l.((P\mathcal{R}Q \wedge P \xrightarrow{l} P') \Rightarrow \exists Q'.(Q \xrightarrow{l} Q' \wedge P' \leq_{sim} \mathcal{R} \leq_{sim} Q')),$$

then $\mathcal{R} \subseteq \leq_{sim}$.

To show the "only if" part of Theorem 2, it suffices to show that the following relation is a simulation up to $\leq_{sim}$ (i.e., satisfies the assumption of Lemma 5).

$$\begin{aligned}[\![\leq_{sim}]\!] = \{&(E,E) \mid E \in \mathbf{BPP}_{\{a,b\}}\} \\ \cup\ \{&([\![P]\!]_L, [\![Q]\!]_{R'}) \mid P \leq_{sim} Q\} \\ \cup\ \{&(E,F) \mid P \leq_{sim} Q \wedge P' \leq_{sim} Q' \wedge s_1 s_2 = [\![l]\!] \wedge s_1, s_2 \neq \epsilon \wedge \\ &[\![P]\!]_L \xrightarrow{s_1} E \xrightarrow{s_2} [\![P']\!]_L \wedge [\![Q]\!]_{R'} \xrightarrow{s_1} F \xrightarrow{s_2} [\![Q']\!]_{R'}\}\end{aligned}$$

E and F in the third set are intermediate states for simulating a single action of general BPP processes. If E performs a valid action and becomes E', then F can also perform a valid action to become F' so that the pair (E', F') is again in the second or third set. If E performs an invalid action to become E', then F can transit to a process containing $H^{(2N-2-k)}$ or G, which can simulate any transitions of E'. See [11] for the full proof.

As a corollary of the above theorem and the undecidability for general BPP [4, 12], we obtain the undecidability for 2-label BPP.

Corollary 2. *The relations $\leq_{sim}$ and $\sim_{sim}$ are undecidable for 2-label BPP.*

5 Application to Behavioral Type Systems

In this section, we apply the undecidability results of the previous sections to show the undecidability of certain behavioral type systems for the π-calculus.

Behavioral type systems [1, 6, 9, 10, 16] use types to control how processes may interact with each other. They have been used for analyzing deadlocks, race conditions, and termination, etc. The version of behavioral type systems we discuss below is a type system with *channel usages* [8–10, 14], which express how each communication channel is used for input and output.

5.1 Syntax of Usages, Types, and Processes

The syntax of usages and types are given by:

$$\begin{aligned}&U \text{ (usages)} ::= \mathbf{0} \mid ?U \mid !U \mid (U_1 \mid U_2) \mid U_1 + U_2 \mid X \mid \mu X.U \\ &\tau \text{ (types)} ::= \mathbf{chan}_U[\tau_1, \ldots, \tau_n]\end{aligned}$$

The syntax of usages is almost the same as that of 2-label BPP, except that X may be unguarded in $\mu X.U$. For example, $\mu X.X$ is allowed and is sometimes distinguished from $\mathbf{0}$ [9, 10]. The transition relation $U \xrightarrow{l} U'$ (where $l \in \{?, !\}$) is the same as that of $\mathbf{BPP}_{\{?,!\}}$. We often omit $\mathbf{0}$ and just write ! and ? for $!\mathbf{0}$ and $?\mathbf{0}$. We write $FV(U)$ for the set of free variables in U.

Table 1 summarizes intuitive meaning of usages. For example, the usage $?\,|\,!$ describes a channel that should be used once for input and once for output in parallel.

Table 1. Intuitive Meaning of Usages

$\mathbf{0}$	not used at all
$?U$	used for input, and then according to U
$!U$	used for output, and then according to U
$U_1 \,\|\, U_2$	used according to U_1 and U_2, possibly in parallel
$U_1 + U_2$	used according to either U_1 or U_2
X	usage variable bound by μ.
$\mu X.U$	used recursively according to $X = U$.

The type $\mathbf{chan}_U[\tau_1, \ldots, \tau_n]$, abbreviated to $\mathbf{chan}_U[\widetilde{\tau}]$, describes a channel that should be used for passing a tuple of channels of types $\tau_1, \ldots, \tau_n$, and used according to U. For example, the type $\mathbf{chan}_{?!}[\,]$ describes a channel that should be first used for receiving, and then for sending a null tuple. A channel of type $\mathbf{chan}_?[\mathbf{chan}_![\,]]$ should be used for receiving a channel, and then the received channel should be used for sending a null tuple.

The subtyping relation $\tau_1 \le \tau_2$ (which means that a value of type τ may be used as a value of type τ') is defined by: $\mathbf{chan}_U[\widetilde{\tau}] \le \mathbf{chan}_{U'}[\widetilde{\tau}]$ if and only if $U \le U'$. Here, the *subusage* relation $U \le U'$ means that U represents a more liberal usage of channels, so that a channel of usage U may be used as a channel of usage U'. For example, $?+! \le ?$ should hold. There are several reasonable definitions of the subusage relation [7–10], depending on the property that should be ensured by the type system. The following definition is the simplest one among such reasonable definitions; other definitions are discussed later.

Definition 7 $U_1 \le U_2 \overset{def}{\Leftrightarrow} U_2 \le_{tr} U_1$.

Here, $\le_{tr}$ is the trace inclusion relation for $\mathbf{BPP}_{\{?,!\}}$.

The syntax of processes is given by:

$$P ::= \mathbf{0} \mid x![y_1, \ldots, y_n].\,P \mid x?[y_1, \ldots, y_n].\,P \mid (P \,|\, Q) \mid *P \mid (\nu x : U)\,P$$

A sequence $y_1, \ldots, y_n$ is abbreviated to $\widetilde{y}$. The process $x![\widetilde{y}].\,P$ sends the tuple $[\widetilde{y}]$ of channels on channel x, and then behaves like P. The process $x?[\widetilde{y}].\,P$ waits to receive a tuple consisting of channels $\widetilde{z}$ on channel x, binds $\widetilde{y}$ to them, and then behaves like P. The process $P \,|\, Q$ runs P and Q in parallel, and the process $*P$ runs infinitely many copies of P in parallel. The process $(\nu x \colon U)\, P$ creates a fresh communication channel, binds x to it, and then behaves like P. An important point here is that x is annotated with a usage U, which specifies a programmer's intention on how x should be used. As observed later, this usage declaration makes the type system described below undecidable. We do not consider choice $P + Q$ and name matching $[x = y]P$; The type system remains undecidable in the presence of those constructors.

Example 1. In the π-calculus, a lock (i.e., a binary semaphore) can be expressed as a channel, where the locked (unlocked, resp.) state is represented by the absence (presence, resp.) of a message. For example, the process $lck?[\,].\, x?[y].\, lck![\,]$ locks lck, reads from x, and then releases lck. To enforce that the channel lck is indeed used as a lock (i.e., the channel is first used for output to initialize the lock, and then used according to ?! an arbitrary number of times), one can declare a usage of lck as $(\nu lck : (!\,|\,\mu X.(\mathbf{0} + (?!\,|\,X))))\, P$. The type system introduced in the next subsection ensures that P uses lck according to the declared usage.

5.2 Type System

A type judgment for processes is of the form $\Gamma \vdash P$, where Γ is a type environment of the form $x_1 : \tau_1, \ldots, x_n : \tau_n$. It means that P behaves as specified by Γ. For example, $x \colon \mathbf{chan}_{?}[\mathbf{chan}_{!}[\,]] \vdash P$ means that P uses x for receiving a channel of type $\mathbf{chan}_{!}[\,]$, and then uses the received channel for sending a null tuple.

Typing rules and related definitions are given in Figure 2.

Remark 1. Although a usage of the form $U_1 + U_2$ does not appear in Figure 2, it can be introduced by rule T-Sub. For example, the process:
$x![y] \,|\, x?[z].\, z![\,] \,|\, x?[z].\, z?[\,].\, \mathbf{0}$ is typed under $x \colon \mathbf{chan}_{!\,|\,?\,|\,?}[\mathbf{chan}_{!+?}[\,]], y \colon \mathbf{chan}_{!+?}[\,]$.

5.3 Undecidability of Type Checking Problem

We show that the problem of deciding whether $\emptyset \vdash P$ holds or not is undecidable. The key observation for the proof is that, given two usages U_1 and U_2, we can construct a process P such that $\emptyset \vdash (\nu x : U_1)\, P$ if and only if $U_1 \leq U_2$. We use show the following key lemma.

Lemma 6. *Let U be a usage and suppose $FV(U) \subseteq \{X_1, \ldots, X_n\}$. Then there exists a process P such that the followings are equivalent for any $U', U_1, \ldots, U_n$.*

1. $U' \leq [U_1/X_1, \ldots, U_n/X_n]U$
2. $x_1 : \mathbf{chan}_{U_\bot}[\mathbf{chan}_{U_1}[\,]], \ldots, x_n : \mathbf{chan}_{U_\bot}[\mathbf{chan}_{U_n}[\,]], r : \mathbf{chan}_{U'}[\,] \vdash P$.

Here, $U_\bot = \mu X.(\mathbf{0}+?X+!X)$.

We obtain the following result as a corollary of Lemma 6 and Corollary 1.

Operation on type environments:

$$(\Gamma_1 \,|\, \Gamma_2)(x) = \begin{cases} (\Gamma_1(x)) \,|\, (\Gamma_2(x)) & \text{if } x \in \mathit{dom}(\Gamma_1) \cap \mathit{dom}(\Gamma_2) \\ \Gamma_1(x) & \text{if } x \in \mathit{dom}(\Gamma_1) \setminus \mathit{dom}(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \in \mathit{dom}(\Gamma_2) \setminus \mathit{dom}(\Gamma_1) \end{cases}$$

where $\mathbf{chan}_{U_1}[\widetilde{\tau}] \,|\, \mathbf{chan}_{U_2}[\widetilde{\tau}] = \mathbf{chan}_{U_1 \,|\, U_2}[\widetilde{\tau}]$

$(*\Gamma)(x) = *(\Gamma(x))$

where $*\mathbf{chan}_U[\widetilde{\tau}] = \mathbf{chan}_{\mu X.(U \,|\, X)}[\widetilde{\tau}]$

Typing:

$$\frac{}{\emptyset \vdash \mathbf{0}} \;(\text{T-Zero}) \qquad \frac{\Gamma \vdash P}{*\Gamma \vdash *P} \;(\text{T-Rep})$$

$$\frac{\Gamma \vdash P \qquad \Delta \vdash Q}{\Gamma \,|\, \Delta \vdash P \,|\, Q} \;(\text{T-Par}) \qquad \frac{\Gamma, x : \tau \vdash P \qquad \tau' \le \tau}{\Gamma, x : \tau' \vdash P} \;(\text{T-Sub})$$

$$\frac{\Gamma, x : \mathbf{chan}_U[\widetilde{\tau}] \vdash P}{\Gamma \vdash (\nu x : U)\, P} \;(\text{T-New}) \qquad \frac{\Gamma \vdash P \qquad U \le \mathbf{0}}{\Gamma, x : \mathbf{chan}_U[\widetilde{\tau}] \vdash P} \;(\text{T-Weak})$$

$$\frac{\Gamma, x : \mathbf{chan}_U[\widetilde{\tau}] \vdash P}{(\Gamma, x : \mathbf{chan}_{!U}[\widetilde{\tau}]) \,|\, \widetilde{y} : \widetilde{\tau} \vdash x![\widetilde{y}].\, P} \;(\text{T-Out}) \qquad \frac{\Gamma, x : \mathbf{chan}_U[\widetilde{\tau}], \widetilde{y} : \widetilde{\tau} \vdash P}{\Gamma, x : \mathbf{chan}_{?U}[\widetilde{\tau}] \vdash x?[\widetilde{y}].\, P} \;(\text{T-In})$$

Fig. 2. A Behavioral Type System

Theorem 3. *The relation $\emptyset \vdash P$ is undecidable.*

Proof. Let U_1, U_2 be usages. By Lemma 6, there exists a process P_1 such that $r : \mathbf{chan}_{U_1}[\,] \vdash P_1$ if and only if $U_1 \le U_2$. Hence, $\emptyset \vdash (\nu r : U_1)\, P_1$ if and only if $U_1 \le U_2$. Since the latter is undecidable, the type checking problem is also undecidable. $\square$

Undecidability results for other definitions of $U_1 \le U_2$ The above undecidability result holds for various other definitions of the subusage relation. For example, let $\le \overset{def}{=} \ge_{sim}$. Since Lemma 6 remains valid, and $U_1 \le_{sim} U_2$ is undecidable, the type checking problem is also undecidable. Here we sketch other definitions of subusage relations for which the type checking problem remains undecidable.

– Define a predicate $U{\downarrow}$ inductively by the rules:

$$\frac{}{\mathbf{0}{\downarrow}} \qquad \frac{U_1{\downarrow} \quad U_2{\downarrow}}{(U_1 \,|\, U_2){\downarrow}} \qquad \frac{U_i{\downarrow}}{(U_1 + U_2){\downarrow}} \qquad \frac{[\mu X.U/X]U{\downarrow}}{\mu X.U{\downarrow}}$$

Then add the condition $U_1{\downarrow} \Rightarrow U_2{\downarrow}$ to the requirement for each element (U_1, U_2) in the simulation relation $\le_{sim}$. Let $\le_{sim}^{ex}$ be the extended simulation relation, and define $U_1 \le U_2$ as $U_2 \le_{sim}^{ex} U_1$.

– Extend the trace set using the above predicate:

$$\mathit{extraces}(U) = \{l_1 \cdots l_n \mid U \xrightarrow{l_1} \cdots \xrightarrow{l_n} U'\} \cup \{l_1 \cdots l_n{\downarrow} \mid U \xrightarrow{l_1} \cdots \xrightarrow{l_n} U'{\downarrow}\}$$

Then define $U_1 \le U_2$ as $\mathit{extraces}(U_2) \subseteq \mathit{extraces}(U_1)$.

– Add a transition $U \xrightarrow{\tau} U'$ by introducing the synchronization rule:

$$\frac{U_1 \xrightarrow{?} U_1' \quad U_2 \xrightarrow{!} U_2'}{U_1 \,|\, U_2 \xrightarrow{\tau} U_1' \,|\, U_2'}$$

Then re-define the trace set, and define $\leq$ as the trace inclusion relation.

Remark 2. The undecidability results above may be disappointing, given that behavioral type systems are useful for checking various properties [1, 6, 9, 10, 16, 17] and that the above type system is one of the simplest forms of behavioral type systems. It should be noted, however, that the source of the undecidability result is the programmer's capability to declare arbitrary usages (by $(\nu x\!:\!U)$). In fact, Kobayashi's type systems for deadlock-freedom and information flow [9, 10] are much more complex, but the type checking problem is decidable (note that they do not allow type declaration). In order to allow declaration of usages as in this paper while keeping the decidability of type checking, we need to restrict the class of usages that can be declared by programmers. For example, type checking is decidable if the class of declared usages is restricted to the class of usages whose trace sets are deterministic Petri net languages [15].

6 Related Work

As already mentioned in Section 1, Hirshfeld [3] showed the undecidability of the trace equivalence for general BPP, and Hüttel [4] extended the result to show undecidability of other equivalence relations (except bisimilarity, which is decidable [2]). They [3, 4] both encode Minsky machines into BPP. Since their encoding uses more than two action labels, their results do not immediately imply the undecidability for 2-label BPP.

Srba [5] proposed a general method for encoding a labeled transition system into a transition system with a *single* label, so that certain properties are preserved by the encoding. His encoding is, however, not applicable to BPP.

A number of behavioral type systems for the π-calculus have been proposed recently for checking various properties including deadlock, race, liveness, termination, and information flow [1, 6, 9, 10, 16, 17]. Usage-based behavioral type systems studied in Section 5 were first proposed in [14] (in a less general form, without full recursion), and have been extended [1, 9, 10]. The undecidability result presented in this paper indicates that explicit usage or type declarations must be restricted in order to make those type systems decidable.

7 Conclusion

We have shown that the trace equivalence and simulation relation for 2-label BPP is undecidable. The undecidability result also implies the undecidability of certain behavioral type systems for the π-calculus.

Acknowledgments. We would like to thank Hans Hüttel for discussions on the undecidability of BPP equivalences. We would also like to thank anonymous reviewers for useful comments.

References

1. S. Chaki, S. Rajamani, and J. Rehof. Types as models: Model checking message-passing programs. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 45–57, 2002.
2. S. Christensen, Y. Hirshfeld, and F. Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 386–396, 1993.
3. Y. Hirshfeld. Petri nets and the equivalence problem. In *Computer Science Logic*, volume 832 of *Lecture Notes in Computer Science*, pages 165–174. Springer-Verlag, 1993.
4. H. Hüttel. Undecidable Equivalence for Basic Parallel Processes. In *Proceedings of TACS94*, volume 789 of *Lecture Notes in Computer Science*, pages 454–464. Springer-Verlag, 1994.
5. Jirí. On the power of labels in transition systems. In *Proceedings of CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 277–291. Springer-Verlag, 2001.
6. N. Kobayashi. TyPiCal: A type-based static analyzer for the pi-calculus. Tool available at `http://www.kb.ecei.tohoku.ac.jp/~koba/typical/`.
7. N. Kobayashi. A type system for lock-free processes. *Information and Computation*, 177:122–159, 2002.
8. N. Kobayashi. Type systems for concurrent programs. In *Proceedings of UNU/IIST 20th Anniversary Colloquium*, volume 2757 of *Lecture Notes in Computer Science*, pages 439–453. Springer-Verlag, 2003.
9. N. Kobayashi. Type-based information flow analysis for the pi-calculus. *Acta Informatica*, 42(4-5):291–347, 2005.
10. N. Kobayashi. A new type system for deadlock-free processes. In *Proceedings of CONCUR 2006*, volume 4137 of *Lecture Notes in Computer Science*, pages 233–247. Springer-Verlag, 2006.
11. N. Kobayashi and T. Suto. Undecidability of 2-label BPP equivalences and behavioural type systems for the π-calculus, 2007. Full version. Available from `http://www.kb.ecei.tohoku.ac.jp/~koba/publications.html`.
12. N. Kobayashi and T. Suto. Undecidability of BPP equivalences revisited, 2007. Available from `http://www.kb.ecei.tohoku.ac.jp/~koba/publications.html`.
13. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
14. E. Sumii and N. Kobayashi. A generalized deadlock-free process calculus. In *Proc. of Workshop on High-Level Concurrent Language (HLCL'98)*, volume 16(3) of *ENTCS*, pages 55–77, 1998.
15. T. Suto and N. Kobayashi. Channel usage declaration for concurrent programming languages. *IPSJ Transaction on Programming*, 2007. to appear (in Japanese).
16. N. Yoshida. Graph types for monadic mobile processes. In *FST/TCS'16*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–387. Springer-Verlag, 1996.
17. N. Yoshida, M. Berger, and K. Honda. Strong normalisation in the pi-calculus. *Information and Computation*, 191(2):145–202, 2004.