Type-Based Verification of Correspondence Assertions for Communication Protocols

Daisuke Kikuchi and Naoki Kobayashi

Graduate School of Information Sciences, Tohoku University
{kikuchi,koba}@kb.ecei.tohoku.ac.jp

Abstract. Gordon and Jeffrey developed a type system for checking correspondence assertions. The correspondence assertions, proposed by Woo and Lam, state that when a certain event (called an "end" event) happens, the corresponding "begin" event must have occurred before. They can be used for checking authenticity in communication protocols. In this paper, we refine Gordon and Jeffrey's type system and develop a polynomial-time type inference algorithm, so that correspondence assertions can be verified fully automatically, without any type annotations. The main key idea that enables polynomial-time type inference is to introduce fractional effects; Without the fractional effects, the type inference problem is NP-hard.

1 Introduction

Woo and Lam [11] introduced the notion of *correspondence assertions* for stating expected authenticity properties formally. The correspondence assertions consist of *begin*-assertions and *end*-assertions, and assert that whenever an end-event occurs, the corresponding begin-event must have occurred before. For example, consider a simple transmit-acknowledgment-handshake protocol, where a process A sends a message to B and waits for an acknowledgment. Let the event that B receives a message from A be a begin-event, and the event that A receives an acknowledgment be an end-event. By checking that the begin event always precedes the end-event, one can verify that whenever A believes that the message has been received by B, the message has indeed been delivered to B.

Gordon and Jeffrey [7] introduced an extension of the π-calculus with correspondence assertions, and proposed a type-and-effect system for checking correspondence assertions. Since well-typed processes satisfy correspondence assertions, the problem of checking correspondence assertions is reduced to the type-checking problem. They further extended the type system to deal with cryptographic primitives [4–6].

In this paper, we refine Gordon and Jeffrey's type system for correspondence assertions and develop a polynomial-time type inference algorithm, so that correspondence assertions can be verified fully automatically without any type annotations (which were necessary for Gordon and Jeffrey's type checking

algorithm [7]). The key idea to enable type inference is to introduce *fractional* effects, which are mappings from events to rational numbers; In Gordon and Jeffrey's type system, effects are multisets of events, or mappings from events to natural numbers. We show that with fractional effects (and with the assumptions that the size of simple types is polynomial in the size of untyped programs and that the size of events is bound by a constant), type inference can be performed in polynomial time, while without fractional effects, the type inference problem is NP-hard (even with the assumption that both the size of simple types and the size of events are bound by a constant).

The rest of this paper is structured as follows. Section 2 introduces $\pi^{\mathbf{CA}}$, Gordon and Jeffrey's calculus *without* type-and-effect annotations. Section 3 introduces a type system with fractional effects. Section 4 describes a polynomial-time type inference algorithm, and also shows that the type inference problem is NP-hard without fractional effects. Section 5 discusses related work and Section 6 concludes.

2 $\pi^{\mathbf{CA}}$: π-Calculus with Correspondence Assertions

In this section, we introduce the language $\pi^{\mathbf{CA}}$, the π-calculus extended with correspondence assertions. The language is essentially the same as Gordon and Jeffrey's calculus [7], except that there are no type annotations in our language.

2.1 Syntax

Definition 1 (processes) The set of processes, ranged over by P, is given by:

$$\begin{aligned} P \text{ (Processes)} &::= \mathbf{0} \mid x![\tilde{y}] \mid x?[\tilde{y}].P \mid (P_1 \mid P_2) \mid *P \mid (\nu x)P \\ &\quad \mid \mathbf{if}\ x = y\ \mathbf{then}\ P\ \mathbf{else}\ Q \mid \mathbf{begin}\ L.P \mid \mathbf{end}\ L.P \\ L \text{ (Event labels)} &::= \langle x_1, \ldots, x_n \rangle \end{aligned}$$

Here, $\tilde{y}$ abbreviates a sequence of names $y_1, \ldots, y_n$. The meta-variables x_i and y_j range over the set $\mathcal{N}$ of names.

The processes **begin** $L.P$ and **end** $L.P$ are special processes for declaring correspondence assertions; **begin** $L.P$ raises a "**begin** L" event and then behaves like P, while **end** $L.P$ raises an "**end** L" event and then behaves like P. An event label L is a sequence of names.

The remaining processes are those of the standard polyadic, asynchronous π-calculus. The process $\mathbf{0}$ is an inaction. The process $x![\tilde{y}]$ sends a tuple of names $\tilde{y}$ on channel x. The process $x?[\tilde{y}].P$ waits to receive a tuple of names on channel x, binds $\tilde{y}$ to them, and then behaves like P. $P_1 \mid P_2$ runs P_1 and P_2 in parallel, while $*P$ runs infinitely many copies of P in parallel. The process $(\nu x)P$ creates a fresh name, binds x to it, and behaves like P. The process **if** $x = y$ **then** P **else** Q behaves like P if x and y are the same name, and behaves like Q otherwise.

The prefixes $x?[\tilde{y}]$ and (νx) bind $\tilde{y}$ and x respectively. We identify processes up to α-conversion. We assume that α-conversion is implicitly applied so that

bound variables are always different from each other and free variables. We often omit trailing $\mathbf{0}$, and write **end** L for **end** $L.\mathbf{0}$.

Example 1. The transmit-acknowledgment-handshake protocol mentioned in Section 1 can be expressed as follows [7]:

$$(\nu c)(\mathit{Sender}(a,b,c) \mid \mathit{Receiver}(a,b,c)),$$

where $\mathit{Sender}(a,b,c)$ and $\mathit{Receiver}(a,b,c)$ are:

$$\mathit{Sender}(a,b,c) \stackrel{\triangle}{=} (\nu msg)(\nu ack)(c![msg, ack] \mid ack?[\,].\mathbf{end}\ \langle a,b,msg\rangle)$$

$$\mathit{Receiver}(a,b,c) \stackrel{\triangle}{=} c?[m,r].\mathbf{begin}\ \langle a,b,m\rangle.r![\,]$$

$\mathit{Sender}(a,b,c)$ creates a fresh message msg, creates a new channel for receiving an acknowledgment, and sends a pair consisting of them on channel c. It then waits for an acknowledgment and raises the "**end** $\langle a,b,msg\rangle$"-event. $\mathit{Receiver}(a,b,c)$ waits to receive a pair $[m,r]$ on channel c, raises a "**begin** $\langle a,b,m\rangle$"-event (where m is bound to msg), and then sends an acknowledgment on r.

As explained in Section 1, the property "whenever *Sender* receives an acknowledgment for msg, *Receiver* has received it" can be captured by the property that whenever an "**end** $\langle a,b,msg\rangle$"-event occurs, a "**begin** $\langle a,b,msg\rangle$"-event must have occurred before.

2.2 Semantics

We give below the operational semantics of $\pi^{\mathbf{CA}}$ and then define the safety of a process, meaning that whenever an end-event occurs, the corresponding begin-event must have occurred before. Note that the semantics is essentially the same as that of Gordon and Jeffrey's calculus.

The operational semantics is defined via the reduction relation $\langle \Psi, E, N\rangle \longrightarrow \langle \Psi', E', N'\rangle$, where Ψ is a multiset of processes, N is a set of names, and E is a multiset consisting of event labels L such that the event **begin** L has been raised but **end** L has not. The reduction relation is defined by the rules in Figure 1.

$$\langle \Psi \uplus \{x?[\tilde{y}].P, x![\tilde{z}]\}, E, N\rangle \longrightarrow \langle \Psi \uplus \{[\tilde{z}/\tilde{y}]P\}, E, N\rangle$$
$$\langle \Psi \uplus \{P \mid Q\}, E, N\rangle \longrightarrow \langle \Psi \uplus \{P, Q\}, E, N\rangle$$
$$\langle \Psi \uplus \{*P\}, E, N\rangle \longrightarrow \langle \Psi \uplus \{*P, P\}, E, N\rangle$$
$$\langle \Psi \uplus \{(\nu x)P\}, E, N\rangle \longrightarrow \langle \Psi \uplus \{[y/x]P\}, E, N \cup \{y\}\rangle \ (y \notin N)$$
$$\langle \Psi \uplus \{\mathbf{if}\ x = y\ \mathbf{then}\ P\ \mathbf{else}\ Q\}, E, N\rangle \longrightarrow \langle \Psi \uplus \{P\}, E, N\rangle \ (\text{if } x = y)$$
$$\langle \Psi \uplus \{\mathbf{if}\ x = y\ \mathbf{then}\ P\ \mathbf{else}\ Q\}, E, N\rangle \longrightarrow \langle \Psi \uplus \{Q\}, E, N\rangle \ (\text{if } x \neq y)$$
$$\langle \Psi \uplus \{\mathbf{begin}\ L.P\}, E, N\rangle \longrightarrow \langle \Psi \uplus \{P\}, E \uplus \{L\}, N\rangle$$
$$\langle \Psi \uplus \{\mathbf{end}\ L.P\}, E \uplus \{L\}, N\rangle \longrightarrow \langle \Psi \uplus \{P\}, E, N\rangle$$

Fig. 1. Operational Semantics

We write $\langle \Psi, E, N \rangle \longrightarrow \mathbf{Error}$ if $\mathbf{end}\ L.P \in \Psi$ but $L \notin E$. We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$. The safety of a process is defined as follows.

Definition 2 (safety) A process P is *safe* if $\langle \{P\}, \emptyset, N \rangle \not\longrightarrow^* \mathbf{Error}$, where N is the set of free names in P.

3 Type System

In this section, we introduce a type-and-effect system for checking the safety of a process. The main differences between our type system and Gordon and Jeffrey's type system [7] (GJ type system, in short) are (i) an effect in our type system is a mapping from event labels to *rational numbers* whereas an effect in GJ type system is a mapping from event labels to natural numbers, and (ii) processes are implicitly-typed in our type system.

3.1 Types and Effects

We first introduce the syntax of types and effects.

Definition 3 (effects) The sets of *types* and *effects*, ranged over by T and e, are given by:

$$\begin{array}{rl} T \text{ (Types)} & ::= \mathbf{Name} \mid \mathbf{Ch}(T_1, \ldots, T_n)e \\ e \text{ (Effects)} & ::= [L_1 \mapsto t_1, \ldots, L_n \mapsto t_n] \\ L \text{ (extended event labels)} & ::= \langle \alpha_1, \ldots, \alpha_k \rangle \\ \alpha \text{ (extended names)} & ::= x \mid \iota \\ \iota \text{ (indices)} & ::= \uparrow \iota \mid 1 \mid 2 \mid \cdots \end{array}$$

Here, $t_1, \ldots, t_n$ ranges over the set of non-negative rational numbers.

Note that an event label has been extended to a sequence of *extended names*. An extended name is either a name (ranged over by $x, y, \ldots$), or an index ι of the form $\uparrow \cdots \uparrow n$.

An effect $[L_1 \mapsto t_1, \ldots, L_n \mapsto t_n]$ denotes the mapping f from the set of events to the set of rational numbers such that $f(L_i) = t_i$ for $i \in \{1, \ldots, n\}$ and $f(L) = 0$ for $L \notin \{L_1, \ldots, L_n\}$.

A type is either the type **Name** of pure names (which are not used as a channel), or a channel type of the form $\mathbf{Ch}(T_1, \ldots, T_n)e$. Here, $\mathbf{Ch}(T_1, \ldots, T_n)e$ is the type of channels that can be used for transmitting tuples consisting of values of types $T_1, \ldots, T_n$ with a *latent effect* e. The latent effect e describes capabilities for raising end-events that are passed from a sender to a receiver through the channel. For example, if x has type $\mathbf{Ch}(\mathbf{Name})[\langle y \rangle \mapsto 2, \langle z, w \rangle \mapsto 1]$, then x can be used for passing one name, and when a communication on x occurs, the capabilities to raise "$\mathbf{end}\ \langle y \rangle$" events twice and an "$\mathbf{end}\ \langle z, w \rangle$" event once are passed from the sender to the receiver (so that $x?[u].\mathbf{end}\ \langle y \rangle.\mathbf{end}\ \langle z, w \rangle.\mathbf{end}\ \langle y \rangle$ is a

valid process). Note that we allow fractional effects. For example, if x has type $\mathbf{Ch}(\,)[\langle y\rangle \mapsto 0.5]$, then a *half* of the capability to raise an **end** $\langle y\rangle$ event is passed each time x is used for communication. The process $x?[\,].\mathbf{end}\ \langle y\rangle$ is therefore invalid, but $x?[\,].x?[\,].\mathbf{end}\ \langle y\rangle$ is valid. Dependencies of a latent effect on transmitted names are expressed by using indices. The type $\mathbf{Ch}(\mathbf{Name}, \mathbf{Name})[\langle 1, 2\rangle \mapsto 1]$, which corresponds to the type $\mathbf{Ch}(x : \mathbf{Name}, y : \mathbf{Name})\langle x, y\rangle$ in GJ type system, describes a channel such that when a pair of names x and y are passed through the channel, a capability to raise an **end** $\langle x, y\rangle$ event is passed. The index constructor $\uparrow$ is used to refer to the name occurring in an outer position. For example, $\uparrow 1$ in the type $\mathbf{Ch}(\mathbf{Name}, \mathbf{Ch}(\,)[\langle\uparrow 1\rangle \mapsto 1])\langle 1, 2\rangle$ refers to the name passed as the first argument (of type **Name**). Note that the type corresponds to $\mathbf{Ch}(x : \mathbf{Name}, y : \mathbf{Ch}(\,)\langle x\rangle)\langle x, y\rangle$ in GJ type system. Thanks to this canonical representation of types, renaming is unnecessary for unification or matching of two types; That is convenient for the type inference algorithm described in Section 4.

A substitution $[x_1/\iota_1, \ldots, x_k/\iota_k]$, denoted by meta-variable θ, is a mapping from indices to names, The substitution, summation, least upper-bound, and order $\leq$ on effects are defined by:

$$
\begin{aligned}
&(\theta e)(L) = \Sigma\{e(L') \mid \theta L' = L\}\\
&(e_1 + e_2)(L) = e_1(L) + e_2(L)\\
&(e_1 \vee e_2)(L) = \mathbf{max}(e_1(L), e_2(L))\\
&e \leq e' \Leftrightarrow \forall L.e(L) \leq e'(L)
\end{aligned}
$$

The substitution θT on types is defined by:

$$
\begin{aligned}
&\theta\mathbf{Name} = \mathbf{Name}\\
&\theta\mathbf{Ch}(T_1, \ldots, T_n)e = \mathbf{Ch}((\uparrow\theta)T_1, \ldots, (\uparrow\theta)T_n)\theta e
\end{aligned}
$$

Here, $\uparrow[x_1/\iota_1, \ldots, x_k/\iota_k]$ denotes $[x_1/\uparrow\iota_1, \ldots, x_k/\uparrow\iota_k]$.

3.2 Typing Rules

We introduce three judgment forms: $\Gamma \vdash \diamond$, meaning that the type environment Γ is well-formed; $N \vdash T$, meaning that the type T is well-formed under the set N of names; and $\Gamma \vdash P : e$, meaning that P has effect e under the type environment Γ. The judgment $N \vdash T$ is used to exclude ill-formed types like $\mathbf{Ch}(\mathbf{Name})[\langle 2\rangle \mapsto 1]$ (which refers to a non-existent index 2). The judgment $\Gamma \vdash P : e$ intuitively means that P may raise end-events described by e that are not preceded by begin-events. In other words, P is a good process on the assumption that P is given the capabilities to raise end-events described by e. For example, $x : \mathbf{Name}, y : \mathbf{Name} \vdash \mathbf{begin}\ \langle x\rangle.\mathbf{end}\ \langle x, y\rangle.\mathbf{end}\ \langle x\rangle : [\langle x, y\rangle \mapsto 1]$ and $x : \mathbf{Ch}(\mathbf{Name})[\langle 1\rangle \mapsto 2] \vdash x?[y].\mathbf{end}\ \langle y\rangle.\mathbf{end}\ \langle y\rangle : [\,]$ are valid judgments. The latter process receives a capability to raise two "**end** $\langle y\rangle$" events through x. On the other hand, $x : \mathbf{Name} \vdash \mathbf{begin}\ \langle x\rangle.\mathbf{end}\ \langle x\rangle.\mathbf{end}\ \langle x\rangle : [\,]$ is an invalid

judgment.

The relations $\Gamma \vdash \diamond$, $N \vdash T$, and $\Gamma \vdash P : e$ are defined by the rules in Figure 2. In the figure, $\theta_{y_1,\ldots,y_k}$ denotes the substitution $[y_1/1, \ldots, y_k/k]$. $\mathbf{N}(e)$ denotes the set $\bigcup\{\mathbf{N}(L) \mid e(L) > 0\}$, where $\mathbf{N}(L)$ is the set of extended names occuring in L. For example, $\mathbf{N}([\langle x, y\rangle \mapsto 0.5, \langle y, z\rangle \mapsto 0]) = \{x, y\}$. The typing rules are basically the same as those of Gordon and Jeffrey's type system [7], except that the syntax of types has been changed and effects have been replaced by mappings from names to rational numbers. In rule WT-Chan, $\uparrow N$ denotes the set $\{\uparrow \iota \mid \iota \in N\} \cup (N \cap \mathcal{N})$. For example, $\uparrow\{x, 1, \uparrow 2\} = \{x, \uparrow 1, \uparrow\uparrow 2\}$. In T-If, $[y/x]\Gamma$ is defined as follows.

$$
\begin{array}{l}
[y/x]\Gamma = [y/x](x_1 : T_1, \ldots, x_n : T_n) = \\
\quad ([y/x]x_1) : ([y/x]T_1); \ldots; ([y/x]x_n) : ([y/x]T_n) \\
\qquad \text{where } \Gamma; x : T \text{ is } \Gamma \text{ if } x \in \mathbf{dom}(\Gamma), \text{ and is } \Gamma, x : T \text{ otherwise}
\end{array}
$$

Example 2. The process $\mathit{Receiver}(a, b, c)$ in Example 1 is typed as follows.

$$
\dfrac{\dfrac{\Gamma_0, m : \mathbf{Name}, r : T_2 \vdash r![\,] : [\langle a, b, m\rangle \mapsto 1]}{\Gamma_0, m : \mathbf{Name}, r : T_2 \vdash \mathbf{begin}\ \langle a, b, m\rangle.r![\,] : [\,]}}{\Gamma_0 \vdash \mathit{Receiver}(a, b, c) : [\,]}
$$

Here, $\Gamma_0 = a : \mathbf{Name}, b : \mathbf{Name}, c : \mathbf{Ch}(\mathbf{Name}, \mathbf{Ch}(\,)[\langle a, b, \uparrow 1\rangle \mapsto 1])[\,]$ and $T_2 = \mathbf{Ch}(\,)[\langle a, b, m\rangle \mapsto 1]$.

Similarly, $\mathit{Sender}(a, b, c)$ is typed as follows.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_2 \vdash c![msg, ack] : [\,] \qquad \dfrac{\Gamma_2 \vdash \mathbf{end}\ \langle a, b, msg\rangle : [\langle a, b, msg\rangle \mapsto 1]}{\Gamma_2 \vdash ack?[\,].\mathbf{end}\ \langle a, b, msg\rangle : [\,]}}{\Gamma_2 \vdash c![msg, ack] \mid ack?[\,].\mathbf{end}\ \langle a, b, msg\rangle : [\,]}}{\Gamma_1 \vdash (\nu ack)(c![msg, ack] \mid ack?[\,].\mathbf{end}\ \langle a, b, msg\rangle) : [\,]}}{\Gamma_0 \vdash \mathit{Sender}(a, b, c) : [\,]}
$$

Here, $\Gamma_1 = \Gamma_0, msg : \mathbf{Name}$ and $\Gamma_2 = \Gamma_1, ack : \mathbf{Ch}(\,)[\langle a, b, msg\rangle \mapsto 1]$. By using T-Par, we obtain:

$$
\Gamma_0 \vdash \mathit{Sender}(a, b, c) \mid \mathit{Receiver}(a, b, c) : [\,].
$$

3.3 Type Soundness

The following theorem states that a process is safe if it is well-typed and has an empty effect.

Theorem 1 (type soundness). *If $\Gamma \vdash P : [\,]$, then P is safe.*

The proof is essentially the same as that of the type sound theorem for GJ type system [7].

$$\frac{}{\emptyset \vdash \diamond} \quad \text{(TE-Empty)}$$

$$\frac{\mathbf{dom}(\Gamma) \vdash T \qquad x \notin \mathbf{dom}(\Gamma)}{\Gamma, x : T \vdash \diamond} \quad \text{(TE-Ext)}$$

$$\frac{}{N \vdash \mathbf{Name}} \quad \text{(WT-Name)}$$

$$\frac{\uparrow(N \cup \{1, \ldots, i-1\}) \vdash T_i \text{ (for each } i \in \{1, \ldots, n\}) \qquad \mathbf{N}(e) \subseteq N \cup \{1, \ldots, n\}}{N \vdash \mathbf{Ch}(T_1, \ldots, T_n)e} \quad \text{(WT-Chan)}$$

$$\frac{\Gamma \vdash P : e \qquad e \leq e' \qquad \mathbf{N}(e') \subseteq \mathbf{dom}(\Gamma)}{\Gamma \vdash P : e'} \quad \text{(T-Subsum)}$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0} : [\,]} \quad \text{(T-Zero)}$$

$$\frac{\Gamma \vdash x : \mathbf{Ch}(T_1, \ldots, T_n)e \qquad \Gamma \vdash y_i : (\uparrow \theta_{y_1,\ldots,y_{i-1}})T_i \text{ (for each } i \in \{1, \ldots, n\})}{\Gamma \vdash x![y_1, \ldots, y_n] : \theta_{y_1,\ldots,y_n} e} \quad \text{(T-Out)}$$

$$\frac{\Gamma \vdash x : \mathbf{Ch}(T_1, \ldots, T_n)e_1 \qquad \Gamma, y_1 : T'_1, \ldots, y_n : T'_n \vdash P : e_2 \qquad T'_i = (\uparrow \theta_{y_1,\ldots,y_{i-1}})T_i \text{ (for each } i \in \{1, \ldots, n\}) \qquad e_2 \leq \theta_{y_1,\ldots,y_n} e_1 + e \qquad (\mathbf{N}(e) \cup \mathbf{N}(e_1)) \cap \{y_1, \ldots, y_n\} = \emptyset}{\Gamma \vdash x?[y_1, \ldots, y_n].P : e} \quad \text{(T-In)}$$

$$\frac{\Gamma \vdash P_1 : e_1 \qquad \Gamma \vdash P_2 : e_2}{\Gamma \vdash P_1 \mid P_2 : e_1 + e_2} \quad \text{(T-Par)}$$

$$\frac{\Gamma \vdash P : [\,]}{\Gamma \vdash *P : [\,]} \quad \text{(T-Rep)}$$

$$\frac{\Gamma, x : T \vdash P : e \qquad x \notin \mathbf{N}(e)}{\Gamma \vdash (\nu x)P : e} \quad \text{(T-Res)}$$

$$\frac{\Gamma \vdash x : T \qquad \Gamma \vdash y : T \qquad [y/x]\Gamma \vdash [y/x]P : [y/x]e_P \qquad \Gamma \vdash Q : e_Q}{\Gamma \vdash \mathbf{if}\ x = y\ \mathbf{then}\ P\ \mathbf{else}\ Q : e_P \vee e_Q} \quad \text{(T-Cond)}$$

$$\frac{\Gamma \vdash P : e + [L \mapsto 1] \qquad \mathbf{N}(L) \subseteq \mathbf{dom}(\Gamma)}{\Gamma \vdash \mathbf{begin}\ L.P : e} \quad \text{(T-Begin)}$$

$$\frac{\Gamma \vdash P : e \qquad \mathbf{N}(L) \subseteq \mathbf{dom}(\Gamma)}{\Gamma \vdash \mathbf{end}\ L.P : e + [L \mapsto 1]} \quad \text{(T-End)}$$

Fig. 2. Typing Rules

3.4 Comparison with GJ Type System

Our type system is *strictly* more expressive than GJ type system [7]. Note that the only difference between our type system and GJ type system is that an effect is a mapping from names to *rational numbers* in our type system, while it is a mapping from names to *natural numbers* in GJ type system. Therefore, it should be trivial that any process well-typed in GJ type system is also well-typed in our type system. On the other hand, there is a process that is typable in our type system but *not* in GJ type system. Consider the following process:[1]

$$\mathbf{begin}\ \langle a\rangle.(c![\,]\,|\,c![\,])\,|\,c?[\,].c?[\,].\mathbf{end}\ \langle a\rangle.$$

The process on the lefthand side first raises a begin-event, and then sends a capability to raise an end-event on channel c, while the process on the righthand side receives the capability from channel c, and raises the end-event.

In our type system, the process is typed under the type environment: $a : \mathbf{Name}, c : \mathbf{Ch}(\mathbf{Name})[\langle a\rangle \mapsto 0.5]$. Note that c carries a *half* of the capability to raise the end-event. Since two messages are sent on c, $0.5 + 0.5 = 1$ capability is passed from the left process to the right process.

To see why the above process is not typable in GJ type system, let the type of c be $\mathbf{Ch}(\,)e$ where $e(\langle a\rangle) = n$. In order for the left process to be typable, it must be the case that $n + n \leq 1$. On the other hand, for the right process to be typable, it must be the case that $n + n \geq 1$. There is no natural number n that satisfies both the constraints.

4 Type Checking Algorithm

This section describes an algorithm which, given a process P, judges whether there exists a type environment Γ such that $\Gamma \vdash P : [\,]$.

The algorithm consists of the following steps.

- Step 1: Generate constraints on effects based on the typing rules.
- Step 2: Reduce the constraints on effects into linear inequalities on rational numbers.
- Step 3: Check whether the linear inequalities have a solution.

We first explain the first and second steps below. We then show in Subsection 4.4 that the algorithm runs in time polynomial in the size of the process (provided that the size of the simple type of each name is polynomial in the size of the process and that the size of each begin/end-event is bound by a constant). In Subsection 4.5, we show that without fractional effects, the type inference problem is NP-hard.

[1] The example was suggested by Tachio Terauchi.

4.1 Step 1: Generating Constraints on Effects

Figure 3 gives an algorithm *Inf*, which takes a closed process P as an input, and generates a set C of constraints. C expresses a necessary and sufficient condition for $\Gamma \vdash P : [\,]$ where all the effects in Γ are empty.

Inf calls a sub-procedure *inf*, which takes a type environment and a process, and generates a pair (e, C) where C is a necessary and sufficient condition for $\Gamma \vdash P : e$. We assume that the simple type of each name has been already inferred by the standard type inference algorithm,[2] and that $\mathbf{typeof}(x)$ decorates the simple type of x with fresh effect variables and returns it. For example, if the simple type of x is $\mathbf{Ch}(\mathbf{Ch}(\mathbf{Name}))$, $\mathbf{typeof}(x)$ returns $\mathbf{Ch}(\mathbf{Ch}(\mathbf{Name})\rho_1)\rho_2$ where ρ_1 and ρ_2 are fresh. In the definition of *inf* (the clauses for input and output processes), $\mathbf{Ch}(T_1, \ldots, T_n)e = \Gamma(x)$ expresses a matching of $\Gamma(x)$ against the pattern $\mathbf{Ch}(T_1, \ldots, T_n)e$. For example, if $n = 1$ and $\Gamma(x) = \mathbf{Ch}(\mathbf{Ch}(\mathbf{Name})\rho_1)\rho_2$, then T_1 and e are instantiated to $\mathbf{Ch}(\mathbf{Name})\rho_1$ and ρ_2 respectively. In the clauses for input and output processes, a substitution θ works as an *operation* for types but as an *constructor* for effects. For example, let $\theta = [a/x, b/1]$. Then $\theta\mathbf{Ch}(\mathbf{Ch}(\mathbf{Name})\rho_1)\rho_2$ is $\mathbf{Ch}(\mathbf{Ch}(\mathbf{Name})[a/x, b/\uparrow 1]\rho_1)[a/x, b/1]\rho_2$. $teq(T_1, T_2)$ matches T_1 and T_2, and generates equality constraints on effects. For example, $teq(\mathbf{Ch}(\mathbf{Ch}(\mathbf{Name})e_1)e_2, \mathbf{Ch}(\mathbf{Ch}(\mathbf{Name})e_1')e_2')$ generates $\{e_1 = e_1', e_2 = e_2'\}$. Note that the number of equality constraints generated by $teq(T_1, T_2)$ is linear in the size of T_1 and T_2. Note also that the matching of types never fails because of the assumption that type inference for the simple type system has been already performed. The substitution operation θT can be performed in time $O(mn)$, where m is the size of T and n is the size of θ. $wf(N, T)$ generates the conditions for T being well-formed under the names N.

Inf generates constraints of the following forms:

- inequalities on effects $e_1 \geq e_2$, where e_1 and e_2 are expressions constructed from effect constants ($[L \mapsto 1]$), effect variables, substitutions, and summation ($+$).
- equalities on effects $e_1 = e_2$, where e_1 and e_2 are either $[\,]$ or of the form $\theta_1 \cdots \theta_k \rho$.
- $\mathbf{notin}(x, e)$, where e is $[\,]$ or an effect variable.
- $\mathbf{N}(e) \subseteq N$, where e is of the form $\theta_1 \cdots \theta_k \rho$.

The algorithm *Inf* is sound and complete with respect to the type system in Section 3 in the following sense.

Lemma 1. *$Inf(P)$ is satisfiable if and only if $\Gamma \vdash P : [\,]$ holds for a type environment Γ such that all the effects in Γ are empty.*

Remark 1. The reason why we require that all the effects in Γ are empty is that P may be executed in parallel with an untrusted process Q. The condition that the effects in Γ are empty ensures that P does not expect to receive any

[2] If there is an undetermined type, let it be **Name**.

capability (to raise end-events) from Q. Thus, even in the presence of the untrusted process Q, the correspondence assertions in P hold. (Since Q may not be simply-typed, execution of $P \mid Q$ may get stuck, however.)

4.2 Step 2: Reducing Constraints on Effects

Let $\mathcal{N}_1$ be the set of all the names occurring in P (including bound names), and let $\mathcal{N}_2$ be the set of indices of the form $\uparrow^k l$. Here, k is less than or equal to the maximum *depth* of the type of a channel occurring in P, and l is less than or equal to the maximum *width* of the type of a channel occurring in P (in other words, the maximum size of tuples sent along channels). Let w be the maximal size of the begin/end-events occurring in P. (Here, the size of $\langle x_1, \ldots, x_k \rangle$ is k.) Then, we need to consider only events in the following set $\mathcal{L}$:

$$\{\langle \alpha_1, \ldots, \alpha_k \rangle \mid k \leq w, \alpha_1, \ldots, \alpha_k \in \mathcal{N}_1 \cup \mathcal{N}_2\}.$$

Note that the size of $\mathcal{N}_1 \cup \mathcal{N}_2$ is polynomial in the size of P, since both the maximum depth and the maximum width of simple types are polynomial in the size of P. On the assumption that the maximum size of events is bound by a constant, therefore, the size N of $\mathcal{L}$ is polynomial in the size of P.

Let $\mathcal{L} = \{L_1, \ldots, L_N\}$. For each effect variable ρ in C, prepare N variables $\xi_{\rho,L_1}, \ldots, \xi_{\rho,L_N}$, ranging over rational numbers. Then, the constraints C on effect variables are replaced with constraints on $\xi_{\rho,L}$ as follows.

$$\begin{aligned}
\mathbf{cconv}(e_1 \geq e_2) &= \{\mathbf{econv}(e_1)(L) \geq \mathbf{econv}(e_2)(L) \mid L \in \mathcal{L}\} \\
\mathbf{cconv}(e_1 = e_2) &= \{\mathbf{econv}(e_1)(L) = \mathbf{econv}(e_2)(L) \mid L \in \mathcal{L}\} \\
\mathbf{cconv}(\mathbf{notin}(x, e)) &= \{\mathbf{econv}(e)(L) = 0 \mid L \in \mathcal{L} \wedge x \in \mathbf{N}(L)\} \\
\mathbf{cconv}(\mathbf{N}(e) \subseteq N) &= \{\mathbf{econv}(e)(L) = 0 \mid L \in \mathcal{L} \wedge \mathbf{N}(L) \not\subseteq N\} \\
\mathbf{econv}(\rho) &= \{L_1 \mapsto \xi_{\rho,L_1}, \ldots, L_N \mapsto \xi_{\rho,L_N}\} \\
\mathbf{econv}(e_1 + e_2) &= \{L_1 \mapsto \mathbf{econv}(e_1)(L_1) + \mathbf{econv}(e_2)(L_1), \ldots, \\
&\qquad L_n \mapsto \mathbf{econv}(e_1)(L_N) + \mathbf{econv}(e_2)(L_N)\} \\
\mathbf{econv}(\theta e) &= \{L_1 \mapsto \Sigma\{\mathbf{econv}(e)(L) \mid \theta L = L_1\}, \ldots, \\
&\qquad L_N \mapsto \Sigma\{\mathbf{econv}(e)(L) \mid \theta L = L_N\}\}
\end{aligned}$$

4.3 Example

Recall the process in Example 1. By the standard type inference, the following types are assigned to names:

$$\begin{aligned}
&a : \mathbf{Name}, b : \mathbf{Name}, c : \mathbf{Ch}(\mathbf{Name}, \mathbf{Ch}(\,)\rho_0)\rho_c, msg : \mathbf{Name}, \\
&ack : \mathbf{Ch}(\,)\rho_{ack}, m : \mathbf{Name}, r : \mathbf{Ch}(\,)\rho_r
\end{aligned}$$

Here, $\rho_0, \rho_c, \rho_{ack}$ are effect variables to express unknown effects.

By running the constraint generation algorithm for $\mathit{Sender}(a, b, c)$, we obtain the following constraints.

$$\begin{aligned}
&\rho_{c!} \geq [msg/1, ack/2]\rho_c, \quad \rho_{ack?} + \rho_{ack} \geq [\langle a, b, msg \rangle \mapsto 1] \\
&\rho_s \geq \rho_{c!} + \rho_{ack?}, \quad \rho_{ack} = [msg/\uparrow 1]\rho_0, \mathbf{notin}(msg, \rho_s), \mathbf{notin}(ack, \rho_s) \\
&\mathbf{N}(\rho_0) \subseteq \{a, b, \uparrow 1\}, \quad \mathbf{N}(\rho_c) \subseteq \{a, b, 1, 2\}, \quad \mathbf{N}(\rho_{ack}) \subseteq \{a, b, c, msg\}
\end{aligned}$$

$$
\begin{array}{l}
\mathit{Inf}(P) = \\
\quad \textbf{let} \\
\qquad \Gamma = \{x : \textbf{typeof}(x) \mid x \in \mathbf{N}(P)\} \\
\qquad (e, C_1) = \mathit{inf}(\Gamma, P) \\
\qquad C_2 = \{e = [\,]\} \\
\qquad\quad \cup \{e_1 = [\,] \mid e_1 \text{ appears in } \Gamma\} \\
\quad \textbf{in } C_1 \cup C_2
\end{array}
$$

$$
\mathit{inf}(\Gamma, \mathbf{0}) = ([\,], \emptyset)
$$

$$
\begin{array}{l}
\mathit{inf}(\Gamma, x![y_1, \ldots, y_n]) = \\
\textbf{let} \\
\quad \textbf{Ch}(T_1, \ldots, T_n)e = \Gamma(x) \\
\quad C = \{\mathit{teq}(\Gamma(y_i), (\uparrow \theta_{y_1,\ldots,y_{i-1}})T_i) \\
\qquad\quad \mid i \in \{1, \ldots, n\}\} \\
\textbf{in } (\rho, C \cup \{\rho \geq \theta_{y_1,\ldots,y_n} e\}) \\
\qquad \text{where } \rho \text{ is fresh}
\end{array}
$$

$$
\begin{array}{l}
\mathit{inf}(\Gamma, x?[y_1, \ldots, y_n].P) = \\
\textbf{let} \\
\quad \Gamma' = \Gamma, \tilde{y} : \textbf{typeof}(\tilde{y}) \\
\quad (e_P, C_P) = \mathit{inf}(\Gamma', P) \\
\quad \textbf{Ch}(T_1, \ldots, T_n)e = \Gamma(x) \\
\quad C_1 = \{\mathit{teq}(\Gamma'(y_i), (\uparrow \theta_{y_1,\ldots,y_{i-1}})T_i) \\
\qquad\quad \mid i \in \{1, \ldots, n\}\} \\
\quad C_2 = \{\textbf{notin}(y_i, \rho) \mid i \in \{1, \ldots, n\}\} \\
\quad C_3 = \{\rho + \theta_{y_1,\ldots,y_n} e \geq e_P\} \\
\quad C_4 = \bigcup_{1 \leq i \leq n} \mathit{wf}(\uparrow(\{y_1, \ldots, y_{i-1}\} \\
\qquad\qquad \cup \textbf{dom}(\Gamma)), \Gamma(y_i)) \\
\textbf{in } (\rho, C_P \cup C_1 \cup C_2 \cup C_3 \cup C_4) \\
\qquad \text{where } \rho \text{ is fresh}
\end{array}
$$

$$
\begin{array}{l}
\mathit{inf}(\Gamma, P \mid Q) = \\
\textbf{let} \\
\quad (e_P, C_P) = \mathit{inf}(\Gamma, P) \\
\quad (e_Q, C_Q) = \mathit{inf}(\Gamma, Q) \\
\textbf{in } (\rho, C_P \cup C_Q \cup \{\rho \geq e_P + e_Q\}) \\
\qquad \text{where } \rho \text{ is fresh}
\end{array}
$$

$$
\begin{array}{l}
\mathit{inf}(\Gamma, *P) = \\
\textbf{let } (e_P, C_P) = \mathit{inf}(\Gamma, P) \\
\textbf{in } ([\,], C_P \cup \{e_P = [\,]\})
\end{array}
$$

$$
\begin{array}{l}
\mathit{inf}(\Gamma, (\nu x)P) = \\
\textbf{let } T = \textbf{typeof}(x) \\
\quad (e_P, C_P) = \mathit{inf}((\Gamma, x : T), P) \\
\textbf{in } (\rho, C_P \cup \{\rho \geq e_P\} \cup \{\textbf{notin}(x, e_P)\} \\
\qquad \cup \mathit{wf}(\textbf{dom}(\Gamma), T)) \\
\qquad \text{where } \rho \text{ is fresh}
\end{array}
$$

$$
\begin{array}{l}
\mathit{inf}(\Gamma, \textbf{if } x = y \textbf{ then } P \textbf{ else } Q) = \\
\textbf{let} \\
\quad (e_P, C_P) = \mathit{inf}([y/x]\Gamma, [y/x]P) \\
\quad (e_Q, C_Q) = \mathit{inf}(\Gamma, Q) \\
\quad C_1 = \{[y/x]\rho \geq e_P, \rho \geq e_Q\} \\
\textbf{in } (\rho, C_P \cup C_Q \cup C_1 \cup \{\mathit{teq}(\Gamma(x), \Gamma(y))\}) \\
\qquad \text{where } \rho \text{ is fresh}
\end{array}
$$

$$
\begin{array}{l}
\mathit{inf}(\Gamma, \textbf{begin } L.P) = \\
\textbf{let } (e_P, C_P) = \mathit{inf}(\Gamma, P) \\
\textbf{in } (\rho, C_P \cup \{\rho + [L \mapsto 1] \geq e_P\}) \\
\qquad \text{where } \rho \text{ is fresh and } \mathbf{N}(L) \subseteq \textbf{dom}(\Gamma)
\end{array}
$$

$$
\begin{array}{l}
\mathit{inf}(\Gamma, \textbf{end } L.P) = \\
\textbf{let } (e_P, C_P) = \mathit{inf}(\Gamma, P) \\
\textbf{in } (\rho, C_P \cup \{\rho \geq e_P + [L \mapsto 1]\}) \\
\qquad \text{where } \rho \text{ is fresh and } \mathbf{N}(L) \subseteq \textbf{dom}(\Gamma)
\end{array}
$$

$$
\mathit{teq}(T, T) = \emptyset
$$

$$
\begin{array}{l}
\mathit{teq}(\textbf{Ch}(\tilde{T})e_1, \textbf{Ch}(\tilde{T'})e_2) = \\
\qquad \{e_1 = e_2\} \cup \mathit{teq}(\tilde{T}, \tilde{T'})
\end{array}
$$

$$
\mathit{wf}(N, \textbf{Name}) = \emptyset
$$

$$
\begin{array}{l}
\mathit{wf}(N, \textbf{Ch}(T_1, \ldots, T_n)e) \\
\quad = (\bigcup_{1 \leq i \leq n} \mathit{wf}(\uparrow(N \cup \{1, \ldots, i-1\}), T_i)) \\
\qquad \cup \{\mathbf{N}(e) \subseteq N \cup \{1, \ldots, n\}\}
\end{array}
$$

Fig. 3. Constraint Generation Algorithm

Here, ρ_s, $\rho_{c!}$, and $\rho_{ack?}$ are effects of the processes $\mathit{Sender}(a,b,c)$, $c![\mathit{msg},\mathit{ack}]$, and $\mathit{ack}?[].\cdots$ respectively. The constraints on the first line come from the output and input processes, and those on the second line come from the parallel composition and ν-prefixes. Those on the third line come from the well-formedness conditions (*wf* in the algorithm).

Similarly, we obtain the following constraints from $\mathit{Receiver}(a,b,c)$:

$$\rho_{bg} + [\langle a,b,m\rangle \mapsto 1] \geq \rho_r, \quad \rho_{rec} + [m/1, r/2]\rho_c \geq \rho_{bg}, \quad \rho_r = [m/\uparrow 1]\rho_0$$
$$\mathbf{notin}(m,\rho_{rec}), \mathbf{notin}(r,\rho_{rec}), \mathbf{N}(\rho_r) \subseteq \{a,b,c,m\}$$

From the entire process $(\nu c)(\mathit{Sender}(a,b,c) \mid \mathit{Receiver}(a,b,c))$, we also obtain:

$$\rho_{sys} \geq \rho_s + \rho_{rec}, \quad \rho_{sys} = [\,]$$

The next step is to reduce the above constraints into linear inequalities. The set $\mathcal{L}$ of relevant events is:

$$\{\langle x_1,x_2,x_3\rangle \mid x_1,x_2,x_3 \in \{a,b,c,\mathit{msg},\mathit{ack},m,r,1,2,\uparrow 1\}\}$$

We prepare a variable $\xi_{i,L}$ for each effect variable ρ_i and event L. (In practice, we can reduce the number of variables by looking at the substitutions and events occurring in the effect constraints.)

We show only inequalities relevant to $\langle a,b,\mathit{msg}\rangle$:

$$\xi_{c!,\langle a,b,msg\rangle} \geq \xi_{c,\langle a,b,\uparrow 1\rangle}, \quad \xi_{ack?,\langle a,b,msg\rangle} + \xi_{ack,\langle a,b,msg\rangle} \geq 1$$
$$\xi_{s,\langle a,b,msg\rangle} \geq \xi_{c!,\langle a,b,msg\rangle} + \xi_{ack?,\langle a,b,msg\rangle}, \quad \xi_{ack,\langle a,b,msg\rangle} = \xi_{0,\langle a,b,\uparrow 1\rangle}$$
$$\xi_{s,\langle a,b,msg\rangle} = 0$$
$$\xi_{bg,\langle a,b,m\rangle} + 1 \geq \xi_{r,\langle a,b,m\rangle}, \quad \xi_{c,\langle a,b,1\rangle} \geq \xi_{bg,\langle a,b,m\rangle}, \quad \xi_{r,\langle a,b,m\rangle} = \xi_{0,\langle a,b,\uparrow 1\rangle}$$
$$\xi_{sys,\langle a,b,msg\rangle} \geq \xi_{s,\langle a,b,msg\rangle} + \xi_{rec,\langle a,b,msg\rangle}, \quad \xi_{sys,\langle a,b,msg\rangle} = 0$$

Additionally, we have the inequality $\xi_{i,L} \geq 0$ for every variable.

The above inequalities have the following solution:

$$\xi_{ack,\langle a,b,msg\rangle} = \xi_{r,\langle a,b,m\rangle} = \xi_{0,\langle a,b,\uparrow 1\rangle} = 1,$$
$$\xi_{c!,\langle a,b,msg\rangle} = \xi_{c,\langle a,b,msg\rangle} = \xi_{c,\langle a,b,1\rangle} = \xi_{ack?,\langle a,b,msg\rangle} = \xi_{bg,\langle a,b,m\rangle}$$
$$= \xi_{rec,\langle a,b,msg\rangle} = \xi_{s,\langle a,b,msg\rangle} = \xi_{sys,\langle a,b,msg\rangle} = 0.$$

Thus, we obtain the following type for each name:

$$a : \mathbf{Name}, b : \mathbf{Name}, c : \mathbf{Ch}(\mathbf{Name}, \mathbf{Ch}(\,)[\langle a,b,\uparrow 1\rangle \mapsto 1])[\,], \mathit{msg} : \mathbf{Name},$$
$$\mathit{ack} : \mathbf{Ch}(\,)[\langle a,b,\mathit{msg}\rangle \mapsto 1], m : \mathbf{Name}, r : \mathbf{Ch}(\,)[\langle a,b,m\rangle \mapsto 1],$$

4.4 Efficiency of the Algorithm

Let $|P|$ be the size of the input P of the algorithm *Inf*. We show that, if the size of simple types is polynomial in $|P|$, and the size of begin/end-events is bounded by a constant, then our algorithm runs in time polynomial in $|P|$. First, the number of the constraints generated by $\mathit{Inf}(P)$ is polynomial in $|P|$. Note here that by the

assumption on the size of simple types, the size of $teq(T_1, T_2)$ is also polynomial in $|P|$: the number of effect equalities generated by $teq(T_1, T_2)$ is linear in the size of T_1, and the size of effect expressions occurring in each equality is also polynomial in $|P|$. Since the size of the event set $\mathcal{L}$ is polynomial in the size of the process, the second step runs in time polynomial in $|P|$ and generates linear inequalities of size polynomial in $|P|$. Since linear inequalities can be solved in polynomial time, the third step can also be performed in polynomial time.

Remark 2. Note that in general, the size of simple types (expressed as terms instead of graphs) can be *exponential* in the size of $|P|$. For example consider the following process:

$$x_0![x_1, x_1] \mid x_1![x_2, x_2] \mid \cdots \mid x_n![x_{n+1}, x_{n+1}]$$

The size of the type of x_0 is exponential in n. We believe, however, that in practice, the maximum size of types depends on what data structures and communication protocols are used in the program, and it is generally independent of the size of the program itself. If the assumption on the size of simple types is not met or if there are recursive types, we can use graph representation of simple types and assign the same effect to the same type node. Then, our algorithm still runs in time polynomial in the size of the program, although the completeness of the inference algorithm would be lost.

4.5 Complexity of GJ Type System

Next, we show that the typability in GJ type system [7] is NP-hard without type annotations (in other words, if fractional effects in our type system are replaced by multisets of events as in GJ type system).

Since the 3-SAT problem is NP-complete, in order to prove NP-hardness, it suffices to show that any instance q of 3-SAT problem can be encoded into a process $SAT2P(q)$ so that the size of $SAT2P(q)$ is polynomial in the size of q, and so that q is satisfiable if and only if $\emptyset \vdash SAT2P(q) : [\,]$ is typable.

Let a 3-SAT problem q be $d_1 \wedge \cdots \wedge d_n$ where d_i is $A_{i1} \vee A_{i2} \vee A_{i3}$ and A_{ij} is either a variable X_k or its negation $\overline{X}_k$. By representing the truth value by 1 and 0 (and assuming that each variable ranges over $\{0, 1\}$), we can encode each disjunction d_i into an integer constraint $f2ic(d_i)$, which is one of the following forms:

$$X + Y + Z \geq 1 \qquad X \leq Y + Z \quad X + Y \leq Z + 1 \quad X + Y + Z \leq 2$$

For example, $X_1 \vee \overline{X_2} \vee X_3$ is expressed by $X_1 + (1 - X_2) + X_3 \geq 1$, which is equivalent to $X_2 \leq X_1 + X_3$.

For each of the above inequalities, define the following processes:

$$\begin{aligned}
P_{X+Y+Z\geq 1} &= c_X?[x].c_Y?[y].c_Z?[z]. \\
&\qquad \textbf{if } x = y \textbf{ then if } y = z \textbf{ then end } \langle x \rangle \textbf{ else 0 else 0} \\
P_{X\leq Y+Z} &= c_Y?[y].c_Z?[z].\textbf{if } y = z \textbf{ then } c_X![y] \textbf{ else 0} \\
P_{X+Y\leq Z+1} &= c_Z?[z].\textbf{begin } \langle z \rangle.(c_X![z] \mid c_Y![z]) \\
P_{X+Y+Z\leq 2} &= (\nu a)\textbf{begin } \langle a \rangle.\textbf{begin } \langle a \rangle.(c_X![a] \mid c_Y![a] \mid c_Z![a])
\end{aligned}$$

Then, for each inequality β, β holds if and only if

$$c_X : \mathbf{Ch}(\mathbf{Name})[\langle 1\rangle \mapsto X], c_Y : \mathbf{Ch}(\mathbf{Name})[\langle 1\rangle \mapsto Y], \\ c_Z : \mathbf{Ch}(\mathbf{Name})[\langle 1\rangle \mapsto Z] \vdash P_\beta : [\,]$$

holds.

For each variable X, let Q_X be the process $(\nu a)\mathbf{begin}\ \langle a\rangle.c_X![a]$. Then, $0 \leq X \leq 1$ if and only if $c_X : \mathbf{Ch}(\mathbf{Name})[\langle 1\rangle \mapsto X] \vdash Q_X$.

For an instance of 3-SAT problem $q = d_1 \wedge \cdots \wedge d_n$, define $\mathit{SAT2P}(q)$ by:

$$\mathit{SAT2P}(d_1 \wedge \cdots \wedge d_n) = \\ (\nu c_{X_1}) \cdots (\nu c_{X_m})(Q_{X_1} \mid \cdots \mid Q_{X_m} \mid P_{\mathit{f2ic}(d_1)} \mid \cdots \mid P_{\mathit{f2ic}(d_n)}).$$

Here, $X_1, \ldots, X_m$ are the variables in q. Then, q is satisfiable if and only if $\emptyset \vdash \mathit{SAT2P}(q) : [\,]$ holds. Since the size of $\mathit{SAT2P}(q)$ is linear in the size of q and the 3-SAT problem is NP-complete, the typability (i.e., the problem of judging whether there exists Γ such that $\Gamma \vdash P : [\,]$ holds) is NP-hard.

Remark 3. Note that the above encoding uses only events of the form $\langle a\rangle$ and types of the form $\mathbf{Ch}(\mathbf{Name})e$ and $\mathbf{Name}$. Thus, the type inference problem is NP-hard even on the assumption that the sizes of simple types and events are bound by a constant.

5 Related Work

As already mentioned, this work is based on Gordon and Jeffrey's type system for correspondence assertions [7]. We have extended effects to *fractional effects* and removed explicit type annotations. The resulting type system is more expressive than their type system, and is more suitable for automatic type inference. Gordon and Jeffrey [4–6] have extended their type system to verify authenticity of security protocols using cryptographic primitives. It is left for future work to check whether type inference algorithms for those type systems can be constructed in a similar manner.

Blanchet [1, 2] studied completely different techniques for checking correspondence assertions in cryptographic security protocols, and implemented protocol verification systems. Like in our type system (and unlike in Gordon and Jeffrey's type systems [4–7]), his systems do not require any annotations (except for correspondence assertions). It is difficult to make fair comparison between our work and his techniques because we have not yet extended the type system to deal with cryptographic primitives. A possible advantage of our type-based approach is that our algorithm runs in polynomial time. On the other hand, there seem to be no guarantee that his systems terminate [1]. A clear advantage of his recent work [2] over the type-based methods is that it can guarantee soundness in the computational model (rather than in the formal model with the perfect encryption assumption).

The idea of using rational numbers in type systems have been proposed by Boyland [3] and Terauchi and Aiken [9, 10]. They used rational numbers (ranging

over $[0, 1]$, rather than $[0, \infty)$ in our type system) to prevent interference of read/write operations on reference cells or channels. Terauchi and Aiken [10] observed that type inference can be performed in polynomial time thanks to the use of rational numbers. A main difference between their system and ours is that effects are mapping from *channel handles* to rational numbers in their system [10], while effects are mapping from *names* to rational numbers in ours. Because of the name-dependent feature of GJ type system, reduction of the typability to linear programming was less trivial.

Gordon and Jeffrey's type system for checking correspondence assertions [7] can be regarded as an instance of the generic type system for the π-calculus [8]. Thus, it would be interesting to extend the idea of this paper to develop a type inference algorithm for the generic type system.

6 Conclusion

We have extended Gordon and Jeffrey's type system by introducing fractional effects, and developed a polynomial-time type inference algorithm. Future work includes implementation of the type inference algorithm and extension of the type system to deal with cryptographic primitives.

7 Acknowledgment

We would like to thank Tachio Terauchi for discussions on this work. We would also like to thank anonymous referees for useful comments.

References

1. B. Blanchet. From Secrecy to Authenticity in Security Protocols. In *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359. Springer-Verlag, 2002.
2. B. Blanchet. Computationally sound mechanized proofs of correspondence assertions. In *20th IEEE Computer Security Foundations Symposium (CSF'07)*, pages 97–111. IEEE, 2007.
3. J. Boyland. Checking interference with fractional permissions. In *Proceedings of SAS 2003*, volume 2694 of *Lecture Notes in Computer Science*, pages 55–72. Springer-Verlag, 2003.
4. A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW 2001)*, pages 145–159. IEEE Computer Society Press, 2001.
5. A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. In *15th IEEE Computer Security Foundations Workshop (CSFW-15)*, pages 77–91, 2002.
6. A. D. Gordon and A. Jeffrey. Typing one-to-one and one-to-many correspondences in security protocols. In *Software Security – Theories and Systems, Mext-NSF-JSPS International Symposium (ISSS 2002)*, volume 2609 of *Lecture Notes in Computer Science*, pages 263–282. Springer-Verlag, 2002.

7. A. D. Gordon and A. Jeffrey. Typing correspondence assertions for communication protocols. *Theoretical Computer Science*, 300:379–409, 2003.
8. A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. *Theoretical Computer Science*, 311(1-3):121–163, 2004.
9. T. Terauchi and A. Aiken. Witnessing side-effects. In *Proceedings of International Conference on Functional Programming*, pages 105–115. ACM, 2005.
10. T. Terauchi and A. Aiken. A capability calculus for concurrency and determinism. In *Proceedings of CONCUR 2006*, volume 4137 of *Lecture Notes in Computer Science*, pages 218–232. Springer-Verlag, 2006.
11. T. Y. Woo and S. S. Lam. A semantic model for authentication protocols. In *RSP: IEEE Computer Society Symposium on Research in Security and Privacy*, pages 178–193, 1993.