並行プログラミング言語へのチャネル使用法宣言の導入

須藤 崇[†] 小林直樹[†]

並行プログラムの検証のために，チャネル使用法表現と呼ばれる通信チャネルの使われ方を表す式を導入した型システムが小林らによって提案されている．その型システムを用いることにより，デッドロックやレースなどの様々な性質を静的に検証することができる．本研究では，小林らの型システムを拡張し，各通信チャネルの使用法をプログラマが宣言できるようにする．これにより，たとえばプログラマがある通信チャネルをセマフォとして用いることを宣言し，プログラムがそれに従うことを型検査によって検証することができる．拡張にあたっての主な課題は，使用法表現の間の部分型関係を判定するアルゴリズムの構築である．使用法表現の間の部分型関係は一般には決定不能であるため，プログラマが宣言できる使用法表現のクラスを制限することにより，部分型関係の判定問題をペトリネットの到達可能性判定問題に帰着する．

Introduction of Channel Usage Declaration for Concurrent Programming Languages

Takashi Suto[†] and Naoki Kobayashi[†]

Kobayashi et al. have proposed advanced type systems for concurrent programs, where channel types are extended with channel usage expressions (usages, in short), expressing how each communication channel is used for input and output. Through automatic inference of the extended channel types, we can verify various properties of concurrent programs such as deadlock and race-freedom. In this paper, we extend the previous type systems to allow a programmer to explicitly declare a channel's usage. Thanks to that extension, a programmer can declare, for example, that a certain channel should be used as a semaphore, and it can be statically checked through type inference that a program indeed obeys the declaration. The main new problem caused by that extension is that type checking requires a decision algorithm for the subusage relation; in other words, an algorithm for deciding whether an inferred usage conforms to a declared usage. The subusage relation is unfortunately undecidable. We, however, show that by restricting the class of declared usages, the subusage relation can be reduced to the reachability of Petri nets. Thus, the subusage relation (hence, type checking also) becomes decidable.

1. はじめに

並行プログラムは，実行の非決定性やデッドロックなどの問題のために，テスト実行によるデバッグが困難であり，型検査などの静的なプログラム検査手法の役割が重要である．しかしながら，既存の並行プログラミング言語の型システムは，並行プログラム特有の複雑な誤りを検出するのに不十分である．通信チャネルを介して複数のプロセスがデータを送受信する計算モデルに基づく言語またはライブラリの場合，既存の言語の型システムは送受信されるデータの型の整合性の判定しか行わない．たとえば，以下のようなサーバプロセスを考える．

```
ping?[reply: string chan].reply!"ok"
```

このプロセスは，ping というチャネルから返信のためのチャネル reply を受け取り，reply に文字列 ok を返信する．多くの言語の型システムでは，

```
ping?[reply: string chan].reply!123
```

のように，誤った種類のデータを送信するという誤りは検出することができるが，

```
ping?[reply: string chan].
    (reply!"ok" | reply!"ok")
```

のように，本来 1 度しか返信すべきでないチャネルに 2 回以上送信するという誤りを検出することができない．同様に，通信チャネルを用いてロックを表現する（受信をロックの獲得，送信をロックの解放と見なす）ことが可能であるが，

† 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University

```
do_critical?[lock: unit chan].
   lock?[ ].<critical section>.lock![ ]
```

と記述すべきところを

```
do_critical?[lock: unit chan].
   lock?[ ].<critical section>.
                    (lock![ ] | lock![ ])
```

のようにロックを 2 度解放したり，解放し忘れたりするプログラムを記述しても，型検査に通ってしまう．

上記の問題点を解決するため，本研究では，小林ら[1),2)] により導入された通信チャネルの使われ方を表す**使用法表現**（*usage*）を用いた型をプログラマが宣言できるようにする．これにより，上記のような誤ったプログラムを型検査により排除することができる．たとえば，最後の例は，

```
do_critical?[lock: unit chan(?.!)].
    lock?[ ].<critical section>.
                    (lock![ ] | lock![ ])
```

のように記述することにより，型エラーとして判定される．ここで，`?.!` の部分が通信チャネル `lock` の使われ方の宣言であり，まず受信に 1 度使用した後に送信に 1 度使用すべきであることを表す．

従来の小林らの研究[3),4)] では，チャネルの使用法表現をデッドロックなどの自動解析のために用いてきたが，使用法表現をプログラマが宣言することは許していなかった．使用法表現の宣言を許す拡張を行うにあたっての主な課題は，使用法表現の間の部分型関係（部分使用法関係）の判定を行うアルゴリズムの構築である．部分使用法関係は，ラベルの種類 2 に制限した BPP[5)] のトレースの包含関係に対応し，決定不能であることが分かっている[6),7)]．そこで，本研究では，プログラマが宣言できる使用法表現のクラスを決定性ペトリネット言語に制限する．これにより，部分使用法関係，ひいては型判定の問題をペトリネットの到達可能性問題に帰着させることができる．

本論文の構成は以下のとおり．2 章および 3 章では，対象言語とその型システムを定義する．4 章では，型検査のアルゴリズムを示す．5 章では，型検査器の実装について述べる．6 章で関連研究について議論し，7 章で結論と今後の課題について述べる．

2. 対 象 言 語

本章では，本研究で扱う対象言語として，π 計算[8),9)] に型宣言を付加して得られる言語を定義する．

2.1 型と使用法表現

定義 2.1（型と使用法表現） 型 σ および使用法表現 U の構文は以下で与えられる．

$$
\begin{array}{lll}
\tau & ::= & \mathbf{bool} \mid [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U \\
\sigma & ::= & \tau \mid \mathbf{proc} \\
U & ::= & 0 \mid \alpha \mid ?.U \mid !.U \mid (U_1 \,|\, U_2) \\
 & & \mid U_1 \,\&\, U_2 \mid \mu\alpha.U
\end{array}
$$

型 **bool** はブール値 **true**，**false** の型である．型 $[\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U$ は型 $\tau_1, \ldots, \tau_n$ の値の組を使用法 U に従って送受信する通信チャネルの型である．**proc** はプロセスの型である．

1 章で述べたように，使用法表現 U は各通信チャネルがどのように用いられるかを表す．使用法表現の 0 は送信にも受信にも使用できないことを表す．使用法表現 α は $\mu\alpha$ によって束縛され，再帰で用いられる変数である．使用法表現 $?.U$ は最初に 1 回受信に使われ，その後は使用法表現 U に従って使用されることを表す．たとえば，$?.0$ は 1 回だけ受信に使われることを表す☆．同様に，使用法表現 $!.U$ は最初に 1 回送信に使われ，その後は使用法表現 U に従って使用されることを表す．使用法表現 $(U_1 \,|\, U_2)$ は U_1 と U_2 に従って並行に使われることを表す．使用法表現 $U_1 \,\&\, U_2$ は U_1 と U_2 のどちらにでも使用できることを表す．使用法表現 $\mu\alpha.U$ は $[\mu\alpha.U/\alpha]U$ に従って再帰的に使用できることを表す．たとえば，使用法表現として $\mu\alpha.(0 \,\&\, (!.0 \,|\, \alpha))$ を持つ通信チャネルは並行に任意の回数だけ送信を行うことができる．

以下では，$?.0$ および $!.0$ をそれぞれ ? および ! と略す．U 中の $\mu\alpha$ によって束縛されていない変数 α を自由変数と呼び，自由変数を含まない使用法表現を**閉じた使用法表現**と呼ぶ．

2.2 プ ロ セ ス

プロセスおよび値の構文を以下のように定義する．

定義 2.2（プロセスと値）

$$
\begin{array}{lll}
A\ (\text{式}) & ::= & P \mid v \\
P\ (\text{プロセス}) & ::= & \mathbf{0} \\
 & & \mid x![v_1, \ldots, v_n].\, P \\
 & & \mid x?[y_1 : \tau_1, \ldots, y_n : \tau_n].\, P \\
 & & \mid (P \,|\, Q) \\
 & & \mid (\nu x : \tau)\, P \\
 & & \mid *P \\
 & & \mid \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q \\
v\ (\text{値}) & ::= & x \mid \mathbf{true} \mid \mathbf{false}
\end{array}
$$

プロセス **0** は何もしない．プロセス $x![v_1, \ldots, v_n].\, P$ は通信チャネル x に値の組 $[v_1, \ldots, v_n]$ を送信し，P

☆ ただし，本論文で与える型システムでは，デッドロックがないことを保証しないため，「たかだか」1 回受信に使われることしか保証されない．

を実行する．プロセス $x?[y_1:\tau_1,\ldots,y_n:\tau_n].P$ は通信チャネル x から型 $\tau_1,\ldots,\tau_n$ の値の組 $[v_1,\ldots,v_n]$ を受け取って $y_1,\ldots,y_n$ を $v_1,\ldots,v_n$ に束縛し，プロセス P を実行する．プロセス $(P\,|\,Q)$ は 2 つのプロセス P と Q を並行に実行する．プロセス $(\nu x:\tau)\,P$ は型 τ を持つチャネル x を新たに生成し，プロセス P を実行する．プロセス $*P$ はプロセス P の無限個の複製を並行に実行する．プロセス **if** v **then** P **else** Q は値 v が真ならばプロセス P を，偽ならばプロセス Q を実行する．

上で重要なのは，生成された通信チャネル x および通信チャネルで受信する値 $y_1,\ldots,y_n$ の型が宣言されている点である．これにより，プログラマが意図する通信チャネルの使用法を宣言することが可能となっている．また，本論文では，プロセス中で宣言される使用法表現を閉じた（自由変数を含まない）もののみに限定する．なお，これ以降はプロセスの末尾の $\mathbf{0}$ を省略し，$x![v].\mathbf{0}$ や $x?[y:\tau].\mathbf{0}$ を $x![v]$，$x?[y:\tau]$ と記す．

例 2.3 1 章でも述べたように，同期のためのロックは，各時点でたかだか 1 つしかメッセージが存在しないような通信チャネルとして表現することができ，ロックの獲得をそのチャネルからの受信，解放をそのチャネルへの送信として表現することができる．ロックを生成するためのサーバプロセスは以下のように表すことができる．

$$*\texttt{newlock}?[r:[[\,]\ \mathbf{chan}_{U_{\texttt{lock}}}]\ \mathbf{chan}_!].\\ (\nu l:[\,]\ \mathbf{chan}_{(!\,|\,U_{\texttt{lock}})})\,(l![\,]\,|\,r![l])$$

ここで，$U_{\texttt{lock}}$ は $\mu\alpha.(0\,\&\,(?.!\,|\,\alpha))$，すなわち獲得をしては解放をするという動作を何回でもしてよいというロックの使用法を表す．上のプロセスは，`newlock` を介して（新しいロックの生成要求として）返信先チャネル r を受け取り，新しいロック l を $(\nu l:\cdots)$ により生成，$l![\,]$ により初期化し，r を介して l を返信するプロセスを表す．
クライアントプロセスは以下のように記述できる．

$$(\nu r:[[\,]\ \mathbf{chan}_{U_{\texttt{lock}}}]\ \mathbf{chan}_{(!\,|\,?)})\\ (\texttt{newlock}![r]\,|\,r?[x:[\,]\ \mathbf{chan}_{U_{\texttt{lock}}}].\,P)$$

$\texttt{newlock}![r]$ によりロック生成要求を送信し，$r?[x:[\,]\ \mathbf{chan}_{U_{\texttt{lock}}}].P$ により，生成されたロック x を受け取って P 中で使用する．型宣言 $x:[\,]\ \mathbf{chan}_{U_{\texttt{lock}}}$ により，P 中で x を確かにロックとして使用することを型検査時に検証することができる．

なお，上の例からも分かるとおり，すべてのチャネル生成および受信プロセスに型宣言を付加するのは実際のプログラミングでは煩雑である．実用上は，プログラマが必要な部分だけ宣言を行い，残りの個所には型検査実行前にシステムが自動的に型変数を割り当てることにより，次章以降で議論する型検査を適用することができる．

3. 型システム

本章では，2 章の言語で記述されたプロセスが宣言どおりに各通信チャネルを用いているかどうかを判定するための型システムを定義する．なお，本論文の主題は型検査アルゴリズムの構築であるため，型システムの健全性の議論は省略する．型システムの健全性については，小林らの線形型システムの健全性[10] の拡張として定式化および証明することができる．

3.1 部分使用法と部分型関係

まず，部分使用法関係 $U_1 \le U_2$ を導入する．$U_1 \le U_2$ は，U_1 が U_2 よりも一般的な使用法であり，U_1 に従って用いるべき通信チャネルを U_2 に従って用いてもよいことを意味する．たとえば !&? は送信または受信に 1 回用いてもよいことを表すので，$!\&? \le !$ および $!\&? \le ?$ が成り立つ．

部分使用法関係を定義するために，使用法表現の遷移関係 $U \xrightarrow{t} U'$ を**図 1** によって定義する．ここで，$t \in \{?,!\}$ である．$U \xrightarrow{t} U'$ は，使用法 U を持つ通信チャネルを t で表されるアクションに用いられた後に U' として用いてもよいことを表す．たとえば，使用法 ?.! は以下のように遷移する．

$$?.! \xrightarrow{?} ! \xrightarrow{!} 0$$

使用法表現 U が表す可能な送受信列の集合

$$\{t_1\ldots t_n \mid U \xrightarrow{t_1} \ldots \xrightarrow{t_n} U_n\}\cup\\ \{t_1\ldots t_n\downarrow \mid U \xrightarrow{t_1} \ldots \xrightarrow{t_n} U_n\downarrow\}$$

を $[\![U]\!]$ と書く．

$$?.U \xrightarrow{?} U \qquad !.U \xrightarrow{!} U$$

$$\frac{U_1 \xrightarrow{t} U_1'}{(U_1\,|\,U_2) \xrightarrow{t} (U_1'\,|\,U_2)} \qquad \frac{U_2 \xrightarrow{t} U_2'}{(U_1\,|\,U_2) \xrightarrow{t} (U_1\,|\,U_2')}$$

$$\frac{U_1 \xrightarrow{t} U_1'}{U_1\,\&\,U_2 \xrightarrow{t} U_1'} \qquad \frac{U_2 \xrightarrow{t} U_2'}{U_1\,\&\,U_2 \xrightarrow{t} U_2'}$$

$$\frac{[\mu\alpha.U/\alpha]U \xrightarrow{t} U'}{\mu\alpha.U \xrightarrow{t} U'}$$

図 1 使用法表現の遷移規則
Fig. 1 Transition rules for channel usages.

$$\frac{}{0\downarrow} \qquad \frac{[\mu\alpha.U/\alpha]U\downarrow}{\mu\alpha.U\downarrow} \qquad \frac{U_1\downarrow \qquad U_2\downarrow}{(U_1\,|\,U_2)\downarrow} \qquad \frac{U_i\downarrow\,(i \text{ is 1 or 2})}{(U_1\,\&\,U_2)\downarrow}$$

図 2 使用法表現に関する公理
Fig. 2 Axioms for channel usage.

ここで，使用法表現に関する 1 項関係 $U\downarrow$ の定義は，**図 2** で与えられる．直観的には，$U\downarrow$ ならば，U は送受信にまったく使用しなくてもよいチャネルを表す．

以上の定義を用い，閉じた使用法表現に関する二項関係 $\leq$ を以下によって定義する．

定義 3.1（部分使用法関係）

$$U_1 \leq U_2 \overset{\text{def}}{\Longleftrightarrow} [\![U_2]\!] \subseteq [\![U_1]\!]$$

たとえば，$[\![!.(!\,\&\,0)]\!] = \{\epsilon, !, !!, !\downarrow, !!\downarrow\}$，$[\![!\,]\!] = \{\epsilon, !, !\downarrow\}$ なので $!.(!\,\&\,0) \leq !$ が成り立つ．一方，$[\![!.!]\!] = \{\epsilon, !, !!, !!\downarrow\}$ なので，$!.! \leq !$ は成り立たない．後者の部分使用法関係を禁止している理由は，!.! が送信に 2 回用いることを要求しているのに対し，! は送信を 1 回しか要求していないためである．

なお，自由変数を含む使用法表現については部分使用法関係が定義されないことに注意されたい．したがって，以下で定義する部分型関係および型判定関係も，自由な使用法変数を含むものについては定義されない．

部分使用法関係を用いて，「型 τ_1 の値を型 τ_2 の値と見なすことができる」ことを表す部分型関係 $\tau_1 \leq \tau_2$ を定義する．

定義 3.2（部分型関係） 部分型関係 $\leq$ は，以下の規則を満たす型の間の最小の 2 項関係である．

$$\mathbf{bool} \leq \mathbf{bool} \quad \text{(UT-SubBool)}$$

$$\frac{U \leq U' \qquad \tau_i \leq \tau_i' \qquad \tau_i' \leq \tau_i (\text{for each } i \in \{1,\ldots,n\})}{[\tau_1,\ldots,\tau_n]\ \mathbf{chan}_U \leq [\tau_1',\ldots,\tau_n']\ \mathbf{chan}_{U'}} \quad \text{(UT-SubChan)}$$

注 3.3 上の部分使用法関係およびそれを用いて定義される部分型関係は，小林のデッドロックの解析のための型システム[3] のものとは少し異なっている．本論文の部分使用法関係は，おおよそ，小林の型システム[3] において各送信，受信アクション（!, ?）に “obligation” 属性がついているものと解釈したものに相当する．また，上の定義（**定義 3.1**）では，使用法表現のトレースを用いて部分使用法関係を定義しているので，$(?\,|\,!)$ と $(?.!)\,\&\,(!.?)$ は等価と判断される．これらを区別したい場合には，使用法表現の遷移ラベルとして内部の通信を表す τ を付加し，

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

という遷移規則を追加する必要がある．そのような拡張をした場合にも後述の部分使用法関係の判定アルゴリズムは適用可能である．

3.2 型判定規則

変数の有限集合から型の集合への写像を**型環境**と呼び，メタ変数 Γ を用いて表す．変数 x_i を τ_i（$i = 1,\ldots,n$）に写像する型環境を $x_1:\tau_1,\ldots,x_n:\tau_n$ と記す．

型判定式は $\Gamma \vdash v:\tau$ および $\Gamma \vdash P:\mathbf{proc}$ の形であり，前者は値のための型判定，後者はプロセスのための型判定を表す．$\Gamma \vdash P:\mathbf{proc}$ はプロセス P が各通信チャネルを Γ に従って用いることを表す．たとえば，$\Gamma = x:[\,]\ \mathbf{chan}_?$ とすると，$\Gamma \vdash x?[\,]$ は正しい型判定だが，$\Gamma \vdash x![\,]$ や $\Gamma \vdash x?[\,].x?[\,]$ は誤った型判定である．

型判定関係は**図 3** に示す型判定規則によって定義される．

図中で用いている型環境の 2 項関係 $\Gamma_1 \leq \Gamma_2$ および型環境の演算は以下のように定義される．

定義 3.4（部分型環境関係） 型環境 Γ_1 と型環境 Γ_2 が次の条件をすべて満たすとき，$\Gamma_1 \leq \Gamma_2$ と書く．

(i) $dom(\Gamma_1) \supseteq dom(\Gamma_2)$
(ii) $\forall x.(x \in dom(\Gamma_2) \Rightarrow \Gamma_1(x) \leq \Gamma_2(x))$
(iii) $\forall x.(x \in dom(\Gamma_1) \backslash dom(\Gamma_2) \Rightarrow \Gamma_1(x) = \mathbf{bool} \vee (\Gamma_1(x) = [\tau_1,\ldots,\tau_n]\ \mathbf{chan}_U \wedge U \leq 0))$

例 3.5 $\Gamma_1 = \{x:[\mathbf{bool}]\ \mathbf{chan}_?, y:[\,]\ \mathbf{chan}_0\}$，$\Gamma_2 = \{x:[\mathbf{bool}]\ \mathbf{chan}_?\}$ とすると，$\Gamma_1 \leq \Gamma_2$ が成り立つ．

定義 3.6 型環境に関する演算 $\Gamma\,|\,\Delta$ および $\omega\Gamma$ を以下によって定義する．

$$(\Gamma\,|\,\Delta)(x) = \begin{cases} (\Gamma(x))\,|\,(\Delta(x)) & \text{if } x \in dom(\Gamma) \cap dom(\Delta) \\ \Gamma(x) & \text{if } x \in dom(\Gamma) \backslash dom(\Delta) \\ \Delta(x) & \text{if } x \in dom(\Delta) \backslash dom(\Gamma) \end{cases}$$

$$\mathbf{bool}\,|\,\mathbf{bool} = \mathbf{bool}$$

$$([\tau_1,\ldots,\tau_n]\ \mathbf{chan}_{U_1})\,|\,([\tau_1,\ldots,\tau_n]\ \mathbf{chan}_{U_2})$$

$$\emptyset \vdash \mathbf{true} : \mathbf{bool} \qquad \text{(T-True)}$$

$$\emptyset \vdash \mathbf{false} : \mathbf{bool} \qquad \text{(T-False)}$$

$$x : \tau \vdash x : \tau \qquad \text{(T-Var)}$$

$$\frac{\Gamma \vdash A : \sigma \qquad \Gamma' \leq \Gamma}{\Gamma' \vdash A : \sigma} \qquad \text{(T-Weak)}$$

$$\emptyset \vdash \mathbf{0} : \mathbf{proc} \qquad \text{(T-Zero)}$$

$$\frac{\Gamma_i \vdash v_i : \tau_i \text{ (for each } i \in \{1, \ldots, n\}) \qquad \Gamma', x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U \vdash P : \mathbf{proc}}{\Gamma_1 \,|\, \cdots \,|\, \Gamma_n \,|\, \Gamma', x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_{!.U} \vdash x![v_1, \ldots, v_n].\, P : \mathbf{proc}} \qquad \text{(T-Out)}$$

$$\frac{\Gamma, x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U, y_1 : \tau_1, \ldots, y_n : \tau_n \vdash P : \mathbf{proc}}{\Gamma, x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_{?.U} \vdash x?[y_1 : \tau_1, \ldots, y_n : \tau_n].\, P : \mathbf{proc}} \qquad \text{(T-In)}$$

$$\frac{\Gamma \vdash P : \mathbf{proc} \qquad \Delta \vdash Q : \mathbf{proc}}{\Gamma \,|\, \Delta \vdash (P \,|\, Q) : \mathbf{proc}} \qquad \text{(T-Par)}$$

$$\frac{\Gamma, x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U \vdash P : \mathbf{proc}}{\Gamma \vdash (\nu x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U)\, P : \mathbf{proc}} \qquad \text{(T-New)}$$

$$\frac{\Gamma \vdash P : \mathbf{proc}}{\omega\Gamma \vdash *P : \mathbf{proc}} \qquad \text{(T-Rep)}$$

$$\frac{\Gamma \vdash v : \mathbf{bool} \qquad \Delta \vdash P : \mathbf{proc} \qquad \Delta \vdash Q : \mathbf{proc}}{\Gamma \,|\, \Delta \vdash \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q\ : \mathbf{proc}} \qquad \text{(T-If)}$$

図 3 使用法付きの型判定規則
Fig. 3 Typing rules for the type system with channel usage.

$$= [\tau_1, \ldots, \tau_n]\mathbf{chan}_{(U_1 \,|\, U_2)}$$

$(\omega\Gamma)(x) = \omega(\Gamma(x))$

$\omega\mathbf{bool} = \mathbf{bool}$

$\omega([\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U) = [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_{\omega U}$

$\omega U = \mu\alpha.(0 \,\&\, (U \,|\, \alpha))$

主な規則を以下に説明する．

規則 (T-Weak) は部分型環境に関するものであり，型環境 Γ のもとで型付けできる式はそれよりも一般的な環境 Γ' のもとで型付けできることを表す．

規則 (T-Out) の Γ_i は，v_i が受信プロセスによってどのように使用されるかを表す．結論の $x![v_1, \ldots, v_n].\, P$ の型環境 $\Gamma_1 \,|\, \cdots \,|\, \Gamma_n \,|\, \Gamma', x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_!$ は，そのような受信プロセスによる $v_1, \ldots, v_n$ も含めた，チャネルの使用法を表す．

規則 (T-In) の仮定部分は，受信後のプロセス P が，x を U に従って用い，他のチャネルを Γ に従って用いることを表す．$x?[y_1 : \tau_1, \ldots, y_n : \tau_n].\, P$ はその前に x を受信に用いるので，全体としては，各チャネルを $\Gamma, x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_{?.U}$ に従って用いる．$y_1, \ldots, y_n$ の使用法はこの型環境には含まれていないが，それは上記の (T-Out) の型環境に含まれている．

図 4 は型環境 $x : [[\,]\ \mathbf{chan}_!]\ \mathbf{chan}_{(! \,|\, ?)}, y : [\,]\ \mathbf{chan}_!$ のもとでの式 $x![y] \,|\, x?[z : [\,]\ \mathbf{chan}_!].\, z![\,]$ の型判定の導出例である．なお，導出木において $\tau = [\,]\ \mathbf{chan}_!$ である．

4. 型検査アルゴリズム

本章では，閉じたプロセス式 P が与えられたときに $\emptyset \vdash P : \mathbf{proc}$ が成り立つか否かを判定する型検査アルゴリズムを与える．アルゴリズムは，以下の 3 つのステップから構成される．

(1) $\emptyset \vdash P : \mathbf{proc}$ の必要十分条件を表す，型に関する制約を求める．

(2) (1) で求めた制約を部分使用法関係の制約の集合 $\{D_1 \leq U_1, \ldots, D_n \leq U_n\}$ に簡約する．ここで，$D_1, \ldots, D_n$ は（プログラマによって宣言された）P 中に現れる使用法表現である．

(3) (2) で得られた部分使用法関係の制約が満たされるか否かを判定する．

なお，上記のアルゴリズムは，$D_1, \ldots, D_n$ の表すトレース集合 $[\![D_1]\!], \ldots, [\![D_n]\!]$ が決定性ペトリネット言語[11),12)]（後述）であるときのみ完全である．D

$$\dfrac{\dfrac{}{y:\tau \vdash y:\tau}\ \dfrac{\dfrac{}{\emptyset \vdash \mathbf{0}:\mathbf{proc}}\ (\text{T-Zero})}{x:[\tau]\ \mathbf{chan}_0 \vdash \mathbf{0}:\mathbf{proc}}\ (\text{T-Weak})}{x:[\tau]\ \mathbf{chan}_!, y:\tau \vdash x![y]:\mathbf{proc}}\ (\text{T-Out}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \mathbf{0}:\mathbf{proc}}\ (\text{T-Zero})}{z:[\,]\ \mathbf{chan}_0 \vdash \mathbf{0}:\mathbf{proc}}\ (\text{T-Weak})}{z:[\,]\ \mathbf{chan}_! \vdash z![\,]:\mathbf{proc}}\ (\text{T-Out})}{x:[\tau]\ \mathbf{chan}_0, z:[\,]\ \mathbf{chan}_! \vdash z![\,]:\mathbf{proc}}\ (\text{T-Weak})}{x:[\tau]\ \mathbf{chan}_?\vdash x?[z:\tau].\, z![\,]:\mathbf{proc}}\ (\text{T-In})$$

$$\dfrac{x:[\tau]\ \mathbf{chan}_!, y:\tau \vdash x![y] \qquad x:[\tau]\ \mathbf{chan}_?\vdash x?[z:\tau].\, z![\,]:\mathbf{proc}}{x:[\tau]\ \mathbf{chan}_{(!\,|\,?)}, y:\tau \vdash x![y]\,|\,x?[z:\tau].\, z![\,]:\mathbf{proc}}\ (\text{T-Par})$$

図 4 型判定の導出例
Fig. 4 An example of type judging.

として任意の使用法表現を許した場合には，健全性のみが成り立つ．一般に，任意の使用法表現の間の部分使用法関係 $U_1 \leq U_2$ はラベルが 2 種類の BPP[5] のプロセス間のトレースの包含関係に相当し，その判定問題は決定不能である[7]．

以下では，各ステップについて順に説明する．

4.1 ステップ 1：制約生成アルゴリズム

型環境 Γ とプロセス P（ただし P 中の自由変数はすべて Γ の定義域に含まれる）を入力として，$\Gamma \vdash P : \mathbf{proc}$ が成り立つための必要十分条件を部分型関係と部分使用法関係の制約として出力するアルゴリズム *CheckP* を**図 5** に示す．閉じたプロセス P が入力として与えられたときは，*CheckP*$(\emptyset, P)$ が充足可能であることが $\emptyset \vdash P : \mathbf{proc}$ が成り立つための必要十分条件である．

P が閉じたプロセスであるとき，*CheckP*$(\emptyset, P)$ の出力は部分型関係 $\tau_1 \leq \tau_2$ および部分使用法関係 $U_1 \leq U_2$ の集合である．さらに，*CheckP* の定義から，$\tau_1 \leq \tau_2$ 中に出現する使用法表現のうち，τ_2 の一番外側に現れるもの（τ_2 が $[\tilde{\tau}]\ \mathbf{chan}_U$ のときの U）以外は，変数または（プログラマによる宣言として）P 中に出現する使用法表現である．これは，*CheckP* の一番目の引数 Γ 中に現れる使用法表現が，変数または P 中に出現する使用法表現のみであることから導かれる．

4.2 ステップ 2：制約の簡約

ステップ 1 で得られた制約の集合 C を部分使用法関係の制約の集合に変換するアルゴリズムを**図 6** に示す．ただし，アルゴリズム中の $\tilde{\tau} \leq \tilde{\tau}'$ は $\tau_1 \leq \tau_1', \ldots, \tau_n \leq \tau_n'$ を表す．

ReduceSubType(C) は必ず停止し，使用法表現のみからなる制約の集合または **Error** を出力する．**Error** が出力される場合には，P は（使用法表現とは関係のない）通常の意味で型付け不可能である．

ステップ 1 の出力の形および，簡約アルゴリズム *ReduceSubType*(C) より，以下の性質が成り立つ．

補題 4.1
$U_1 \leq U_2 \in$ *ReduceSubType*(*CheckP*$(\emptyset, P)$) ならば，U_1 は変数または P 中に出現する使用法表現である．

上記補題により，*ReduceSubType*(*CheckP*$(\emptyset, P)$) は，以下の形に表すことができる．

$$\{\alpha_1 \leq U_1, \ldots, \alpha_m \leq U_m, D_1 \leq U_1', \ldots, D_n \leq U_n'\}$$

ここで，$D_1, \ldots, D_n$ は P 中で宣言された使用法表現である．また，$\alpha \leq U_1$ かつ $\alpha \leq U_2$ は $\alpha \leq U_1 \,\&\, U_2$ と等価であること，および制約 $\alpha \leq \alpha$ を追加しても充足可能性が変わらないことから，(i) $\alpha_1, \ldots, \alpha_m$ は互いに異なり，かつ (ii) $\{\alpha_1, \ldots, \alpha_m\}$ は上記の制約に出現する自由変数の集合に一致する，と仮定することができる．したがって，付録の補題 A.1.2 に基づいて，

$$C \cup \{\alpha \leq U\} \longrightarrow [\mu\alpha.U/\alpha]C$$

という書き換えを繰り返し行うことにより，

$$\{D_1 \leq U_1'', \ldots, D_n \leq U_n''\}$$

という形の閉じた使用法表現間の部分使用法関係の制約に変換できる．

4.3 ステップ 3：部分使用法関係の判定アルゴリズム

前のステップで得られた部分使用法関係の制約 $D \leq U$ が成り立つか否かを判定するためのアルゴリズムを示す．

判定アルゴリズムは以下の部分ステップからなる．

- ステップ 3-1：$[\![D]\!]$ および $[\![U]\!]$ を受理するペトリネット $N(D)$ および $N(U)$ を構成する．
- ステップ 3-2：$N(D)$ と $N(U)$ を同時に模倣するペトリネット $N(D) \,||\, N(U)$ を構成する．
- ステップ 3-3：$N(D) \,||\, N(U)$ が「$N(U)$ ができる遷移を $N(D)$ ができない状態」に到達可能であるかを判定する．そのような状態に到達可能でなければ $D \leq U$ が成り立つ．

ただし，$N(D)$ が（後で定義する）決定性ペトリネットである場合のみ，上記のアルゴリズムは完全である．

4.3.1 使用法表現に対応するペトリネットの構成

まず，本論文で扱う（ラベル付き）ペトリネットと

$$
\begin{array}{lcl}
\mathit{CheckP}(\Gamma, \mathbf{0}) & = & \mathit{TopEnv}(\Gamma) \\
\mathit{CheckP}(\Gamma, x![v_1, \ldots, v_n].\, P) & = & C_p \cup C_1 \cup \ldots C_n \cup \mathit{SubEnv}(\Gamma, \Gamma_1 \mid \ldots \mid \Gamma_n \mid \Gamma' \mid (x : [\tilde{\tau}]\ \mathbf{chan}_{!.\alpha})) \\
 & & \quad \mathit{where}\ \Gamma(x) = [\tilde{\tau}]\ \mathbf{chan}_U \\
 & & \qquad \Gamma_i = \mathit{ReNameEnv}(\Gamma)\ (\text{for each } i) \\
 & & \qquad \Gamma' = \mathit{ReNameEnv}(\Gamma) \backslash \{x\} \\
 & & \qquad (C_i, \tau_i) = \mathit{CheckV}(\Gamma_i, v_i) \\
 & & \qquad C_p = \mathit{CheckP}((\Gamma', x : [\tilde{\tau}]\ \mathbf{chan}_\alpha), P) \\
\mathit{CheckP}(\Gamma, x?[y_1 : \tau_1, \ldots, y_n : \tau_n].\, P) & = & C_p \cup \{\Gamma(x) \leq [\tilde{\tau}]\ \mathbf{chan}_{?.\alpha}\} \\
 & & \quad \mathit{where}\ \Gamma' = \Gamma \backslash \{x\}, y_1 : \tau_1, \ldots, y_n : \tau_n, x : [\tilde{\tau}]\ \mathbf{chan}_\alpha \\
 & & \qquad C_p = \mathit{CheckP}(\Gamma', P)) \\
\mathit{CheckP}(\Gamma, (P \mid Q)) & = & C_p \cup C_q \cup \{\Gamma \leq \Gamma_p \mid \Gamma_q\} \\
 & & \quad \mathit{where} \quad C_p = \mathit{CheckP}(\Gamma_p, P), C_q = \mathit{CheckP}(\Gamma_q, Q) \\
 & & \qquad \Gamma_p, \Gamma_q = \mathit{ReNameEnv}(\Gamma) \\
\mathit{CheckP}(\Gamma, (\nu x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U)\, P) & = & \mathit{CheckP}((\Gamma, x : [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U), P) \\
\mathit{CheckP}(\Gamma, *P) & = & \mathit{CheckP}(\Gamma', P) \cup \{\Gamma \leq \omega\Gamma'\} \quad (\mathit{where}\ \Gamma' = \mathit{ReNameEnv}(\Gamma)) \\
\mathit{CheckP}(\Gamma, \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q\ ) & = & C_v \cup C_p \cup C_q \cup \{\Gamma \leq \Gamma_v \mid \Gamma_p, t_v \leq \mathbf{bool}, \mathbf{bool} \leq t_v\} \\
 & & \quad \mathit{where} \quad (C_v, t_v) = \mathit{CheckV}(\Gamma_v, v), C_p = \mathit{CheckP}(\Gamma_p, P), \\
 & & \qquad C_q = \mathit{CheckP}(\Gamma_p, Q), \\
 & & \qquad \Gamma_v, \Gamma_p = \mathit{ReNameEnv}(\Gamma) \\
\mathit{CheckV}(\Gamma, \mathbf{true}) & = & (\mathit{TopEnv}(\Gamma), \mathbf{bool}) \\
\mathit{CheckV}(\Gamma, \mathbf{false}) & = & (\mathit{TopEnv}(\Gamma), \mathbf{bool}) \\
\mathit{CheckV}(\Gamma, x) & = & (\mathit{TopEnv}(\Gamma \backslash \{x\}), \Gamma(x)) \\
 & & \\
\mathit{ReNameEnv}(\Gamma) & = & \{x : \mathit{ReNameType}(\tau) \mid (x : \tau \in \Gamma)\} \\
\mathit{ReNameType}(\mathbf{bool}) & = & \mathbf{bool} \\
\mathit{ReNameType}([\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U) & = & [\tau_1, \ldots, \tau_n]\ \mathbf{chan}_\alpha \\
 & & \text{where } \alpha \text{ is fresh} \\
\mathit{SubEnv}(\Gamma_1, \Gamma_2) & = & \{\Gamma_1(x) \leq \Gamma_2(x) \mid x \in \mathit{dom}(\Gamma_1)\} \\
 & & \\
\mathit{TopEnv}(\Gamma) & = & \bigcup_{x \in \mathit{dom}\Gamma} \mathit{TopType}(\Gamma(x)) \\
\mathit{TopType}(\mathbf{bool}) & = & \emptyset \\
\mathit{TopType}([\tau_1, \ldots, \tau_n]\ \mathbf{chan}_U) & = & \{U \leq 0\}
\end{array}
$$

図 5 型判定のための関数
Fig. 5 Functions for type judging.

その遷移，受理言語を以下のように定義する．

定義 4.2（ペトリネット） **ペトリネット**とは，以下の 4 つ組 (P, P_a, T, I) である．

- P：プレースの集合
- $P_a (\subseteq P)$：受理プレースの集合
- $T (\subseteq (P \to \mathbf{Nat}) \times \{!, ?\} \times (P \to \mathbf{Nat}))$：遷移規則の集合
- $I (\in P \to \mathbf{Nat})$：初期状態

ペトリネット $N = (P, P_a, T, I)$ の状態遷移を表す関係 $m_1 \xrightarrow{t}_N m_2$ を，以下によって定義する．

$$
\begin{array}{l}
m_1 \xrightarrow{t}_N m_2 \overset{\mathtt{def}}{\Longleftrightarrow} \\
\exists m_1', m_2', m_3 \in P \to \mathbf{Nat}. \\
((m_1', t, m_2') \in T \wedge \\
\quad m_1 = m_1' + m_3 \wedge m_2 = m_2' + m_3)
\end{array}
$$

ある $t_1, \ldots, t_n$ について $m_1 \xrightarrow{t_1}_N \cdots \xrightarrow{t_n}_N m_2$ が成り立つとき，$m_1 \Longrightarrow_N m_2$ と書く．特に，$I \Longrightarrow_N m$ が成り立つときに，状態 m は N によって**到達可能**であるという．

ペトリネット N の**受理言語** $\mathbf{L}(N)$ を以下によって定

$$
\begin{array}{lcl}
\mathit{ReduceSubType}(C) & = & C \text{ if } C \text{ 中の制約が部分使用法関係のみ} \\
\mathit{ReduceSubType}(C \uplus \{\mathbf{bool} \leq \mathbf{bool}\}) & = & \mathit{ReduceSubType}(C) \\
\mathit{ReduceSubType}(C \uplus \{[\tilde{\tau}]\ \mathbf{chan}_U \leq [\tilde{\tau}']\ \mathbf{chan}_{U'}\}) & = & \mathit{ReduceSubType}(C \uplus \{\tilde{\tau} \leq \tilde{\tau}', \tilde{\tau}' \leq \tilde{\tau}, U \leq U'\}) \\
\mathit{ReduceSubType}(C \uplus \{\mathbf{bool} \leq [\tilde{\tau}]\ \mathbf{chan}_U\}) & = & \mathbf{Error} \\
\mathit{ReduceSubType}(C \uplus \{[\tilde{\tau}]\ \mathbf{chan}_U \leq \mathbf{bool}\}) & = & \mathbf{Error}
\end{array}
$$

図 6 部分型関係の簡約アルゴリズム
Fig. 6 An algorithm for reducing subtype constrations.

$$
\begin{array}{lcl}
P(U) & = & \mathit{PlcOfUse}(U) \\
P_a(U) & = & \{U' \in P(U) \mid U'\downarrow\} \\
T(U) & = & \bigcup_{U' \in P(U)} \mathit{TrsOfUse}(U') \\
I(U) & = & \mathit{TknToPlc}(U)
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{PlcOfUse}(0) & = & \emptyset \\
\mathit{PlcOfUse}(\alpha) & = & \{\alpha\} \\
\mathit{PlcOfUse}(?.U) & = & \{?.U\} \cup \mathit{PlcOfUse}(U) \\
\mathit{PlcOfUse}(!.U) & = & \{!.U\} \cup \mathit{PlcOfUse}(U) \\
\mathit{PlcOfUse}((U_1 \mid U_2)) & = & \mathit{PlcOfUse}(U_1) \cup \mathit{PlcOfUse}(U_2) \\
\mathit{PlcOfUse}(U_1 \,\&\, U_2) & = & \{U_1 \,\&\, U_2\} \cup \mathit{PlcOfUse}(U_1) \cup \mathit{PlcOfUse}(U_2) \\
\mathit{PlcOfUse}(\mu\alpha.U) & = & \{\mu\alpha.U\} \cup \{[\mu\alpha.U/\alpha]U_1 \mid U_1 \in \mathit{PlcOfUse}(U)\}
\end{array}
$$

$$
\mathit{TknToPlc}(U) = \begin{cases} \emptyset & \text{if } U = 0 \\ \mathit{TknToPlc}(U_1) + \mathit{TknToPlc}(U_2) & \text{if } U = U_1 \mid U_2 \\ \{U \mapsto 1\} & \text{otherwise} \end{cases}
$$

$$
\begin{array}{lcl}
\mathit{Diff}(0, t) & = & \emptyset \\
\mathit{Diff}(\alpha, t) & = & \emptyset \\
\mathit{Diff}(?.U, t) & = & \begin{cases} \{U\} & (\text{if } t = ?) \\ \emptyset & (\text{if } t = !) \end{cases} \\
\mathit{Diff}(!.U, t) & = & \begin{cases} \emptyset & (\text{if } t = ?) \\ \{U\} & (\text{if } t = !) \end{cases} \\
\mathit{Diff}((U_1 \mid U_2), t) & = & \{(U_1' \mid U_2) \mid U_1' \in \mathit{Diff}(U_1, t)\} \cup \{(U_1 \mid U_2') \mid U_2' \in \mathit{Diff}(U_2, t)\} \\
\mathit{Diff}(U_1 \,\&\, U_2, t) & = & \mathit{Diff}(U_1, t) \cup \mathit{Diff}(U_2, t) \\
\mathit{Diff}(\mu\alpha.U, t) & = & \{[\mu\alpha.U/\alpha]U_1 \mid U_1 \in \mathit{Diff}(U, t)\}
\end{array}
$$

$$
\mathit{TrsOfUse}(U) = \{(\mathit{TknToPlc}(U) \xrightarrow{t} \mathit{TknToPlc}(U')) \mid t \in \{!, ?\}, U' \in \mathit{Diff}(U, t)\}
$$

図 7 使用法表現に対応するペトリネットを作成する関数
Fig. 7 Functions which creates petrinet for usages.

義する．

$$
\begin{aligned}
\mathbf{L}(N) = \{&t_1 \cdots t_n \mid I \xrightarrow{t_1}_N \cdots \xrightarrow{t_n}_N m\} \\
&\cup \{t_1 \cdots t_n\downarrow \mid I \xrightarrow{t_1}_N \cdots \xrightarrow{t_n}_N m \wedge \\
&\forall p \in P \backslash P_a.m(p) = 0\}
\end{aligned}
$$

上の定義で P_a は，受理プレースという特別なプレースの集合である．本論文では，すべてのトークンが受理プレースにある状態をペトリネットの受理状態と見なす．

使用法表現に対応するペトリネットを以下のように定義する．

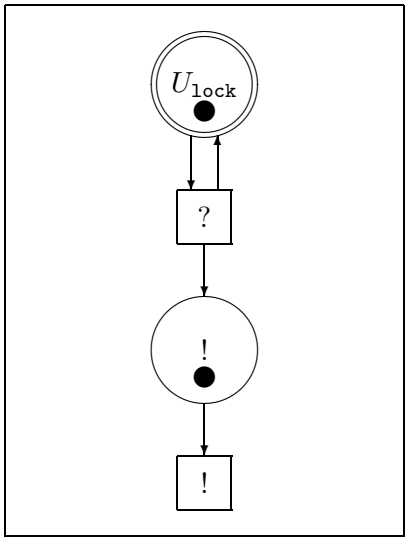

図 8 $!\,|\,U_{\mathtt{lock}}$ のペトリネットでの表現
Fig. 8 A petrinet for a $!\,|\,U_{\mathtt{lock}}$.

定義 4.3（使用法表現に対応するペトリネット） 使用法表現 U に対応するペトリネット $N(U)$ は $(P(U), P_a(U), T(U), I(U))$ の 4 項組である．ここで，$P(U)$，$P_a(U)$，$T(U)$，$I(U)$ は**図 7** によって定義される．

例 4.4 $U = (!\,|\,U_{\mathtt{lock}}) = (!\,|\,\mu\alpha.(0\,\&\,(?.!\,|\,\alpha)))$ に対応するペトリネットは以下の 4 つ組である．

$$P((!\,|\,U_{\mathtt{lock}})) = \{U_{\mathtt{lock}}, 0\,\&\,(?.!\,|\,U_{\mathtt{lock}}),\ ?.!,\ !\}$$
$$P_a((!\,|\,U_{\mathtt{lock}})) = \{U_{\mathtt{lock}},\ 0\,\&\,(?.!\,|\,U_{\mathtt{lock}})\}$$
$$T((!\,|\,U_{\mathtt{lock}})) =$$
$$\{(\{U_{\mathtt{lock}} \mapsto 1\}, ?, \{U_{\mathtt{lock}} \mapsto 1,\ ! \mapsto 1\}),$$
$$(\{0\,\&\,(?.!\,|\,U_{\mathtt{lock}}) \mapsto 1\}, ?, \{U_{\mathtt{lock}} \mapsto 1, ! \mapsto 1\}),$$
$$(\{?.! \mapsto 1\}, ?, \{! \mapsto 1\}), (\{! \mapsto 1\}, !, \emptyset)\}$$
$$I((!\,|\,U_{\mathtt{lock}})) = \{! \mapsto 1,\ U_{\mathtt{lock}} \mapsto 1\}$$

図示すると**図 8** のようになる．ただし，$0\&(?.!\,|\,U_{\mathtt{lock}})$ および $?.!$ は重要ではないので図では省略している．図 8 において白円で表されるのがプレース，四角形で表されているのがトランジション，黒円で表されているのがトークンである．また，受理プレースは二重円で表される．

上の定義より，$[\![U]\!]$ と $\mathbf{L}(N(U))$ は一致する．

補題 4.5 $\mathbf{L}(N(U)) = [\![U]\!]$

したがって，$D \le U$ を判定するためには，$\mathbf{L}(N(U)) \subseteq \mathbf{L}(N(D))$ が成り立つか否かを判定すればよい．

$D \le U$ を決定可能にするため，本論文では，$N(D)$ が以下で定義される決定性ペトリネットであるという条件を課す．

定義 4.6（決定性ペトリネット） ペトリネット $N = (P, P_a, T, I)$ が決定性ペトリネットであるとは，I から到達可能な任意の状態 m において，(m_1, t, m_1'), $(m_2, t, m_2') \in T$ かつ $m \ge m_1$, m_2 ならば，$m_1 = m_2$ かつ $m_1' = m_2'$ が成り立つことをいう．

上の条件は，直観的には，到達可能な任意の状態において，同一のアクションによって到達できる遷移先がただ 1 つに定まることを表す．たとえば，例 4.4 で示したロックの使用法に対応するペトリネットは，決定性ペトリネットである．一方，$?\,\&\,(?.!)$ に対応するペトリネットは決定性ペトリネットではない．

注 4.7 $N(D)$ が決定性ペトリネットであるという条件は，使用法表現 D が以下の条件を満たすことと等価である．

$$\forall U_1, U_2. \forall t_1, \ldots, t_n \in \{!, ?\}.$$
$$(D \xrightarrow{t_1} \cdots \xrightarrow{t_n} U_1 \wedge D \xrightarrow{t_1} \cdots \xrightarrow{t_n} U_2 \Rightarrow U_1 \equiv U_2)$$

ここで，$U_1 \equiv U_2$ は，$U\,|\,0 \equiv U$，$|$ の交換律および結合律，および $\mu\alpha.U \equiv [\mu\alpha.U/\alpha]U$ を満たす最小の関係とする．ただし，D からの到達可能状態 U_1，U_2 は無限に存在しうるので，上の条件を検査するためにはいずれにしても D をペトリネットに変換する必要があると思われる．なお，$D \le U$ を決定可能にするためには，$N(D)$ 自身が決定性ペトリネットであることは必要条件ではなく，$[\![D]\!] = \mathbf{L}(N_1)$ を満たす何らかの決定性ペトリネット N_1 が D から構成できればよい．ただし，上で与えた $N(D)$ より優れた一般的な構成方法は現時点で分かっていない．

4.3.2 ステップ 3-2：ペトリネットの合成

2 つのペトリネット N_1 と N_2 を合成したペトリネット $N_1 \,||\, N_2$ を次のように定義する．

定義 4.8 2 つのペトリネットの合成

2 つのペトリネット $N_1 = (P_1, P_{a1}, T_1, I_1)$ と $N_2 = (P_2, P_{a2}, T_2, I_2)$ を合成したペトリネット $N_1 \,||\, N_2$ を次式で定義する．

$$P = \{(1,p) \mid p \in P_1\} \cup \{(2,p) \mid p \in P_2\}$$
$$P_a = \{(1,p) \mid p \in P_{a1}\} \cup \{(2,p) \mid p \in P_{a2}\}$$
$$T = \{((m_1 \uplus m_2) \xrightarrow{t} (m_1' \uplus m_2')) \mid$$
$$(m_1 \xrightarrow{t} m_1') \in T_1,$$
$$(m_2 \xrightarrow{t} m_2') \in T_2\}$$
$$I = I_1 \uplus I_2$$

ただし，状態 m_1，m_2 の和 $m_1 \uplus m_2$ は $(m_1 \uplus m_2)(1,p) = m_1(p)$，$(m_1 \uplus m_2)(2,p) = m_2(p)$ によって定義されるものとする．

4.3.3 ステップ 3-3：部分使用法関係問題の合成ペトリネットの到達可能性問題への帰着

最後に，決定性ペトリネット N_1 とペトリネット N_2 の受理言語の包含関係 $\mathbf{L}(N_2) \subseteq \mathbf{L}(N_1)$ がステップ 3-2 で合成したペトリネットの到達可能性問題に帰着できることを示す．これにより，部分使用法関係 $D \leq U$ がペトリネットの到達可能性問題に帰着される．

補題 4.9 $N_1 = (P_1, P_{a1}, T_1, I_1)$，$N_2 = (P_2, P_{a2}, T_2, I_2)$ をペトリネットとし，$N = N_1 \parallel N_2 = (P, P_a, T, I)$ とする．N が到達可能なすべての状態 m が次の 2 つの条件を満たせば，$\mathbf{L}(N_2) \subseteq \mathbf{L}(N_1)$ が成り立つ．

(1) $\forall t \in \{!, ?\}.\mathit{Enabled}_2(t, m) \Rightarrow \mathit{Enabled}_1(t, m)$

(2) $\mathit{Accepting}_2(m) \Rightarrow \mathit{Accepting}_1(m)$

ただし，$\mathit{Enabled}_i(t, m)$ は，ある m_1，m_1' について $(m_1, t, m_1') \in T_i$ かつ $\forall p \in P_i.m(i, p) \geq m_1(p)$ が成り立つことを，$\mathit{Accepting}_i(m)$ は，$\forall p \in P_i \backslash P_{ai}.m(i, p) = 0$ が成り立つことを表す．

証明： 任意の $s \in \mathbf{L}(N_2)$ について $s \in \mathbf{L}(N_1)$ が成り立つことを，s の長さに関する帰納法により示す．$s = \epsilon$ のときは明らか．$s = t_1 \cdots t_n t_{n+1}$ のとき．ある m_2 が存在して，$I_2 \xrightarrow{t_1}_{N_2} \cdots \xrightarrow{t_n}_{N_2} m_2 \xrightarrow{t_{n+1}}_{N_2}$ が成り立つ．また，$s \in \mathbf{L}(N_2)$ と $\mathbf{L}(N_2)$ の定義より，$t_1 \cdots t_n \in \mathbf{L}(N_2)$ が成り立つ．したがって，帰納法の仮定より，$t_1 \cdots t_n \in \mathbf{L}(N_1)$．ゆえに，ある m_1 が存在して，N_1 の遷移 $I_1 \xrightarrow{t_1}_{N_1} \cdots \xrightarrow{t_n}_{N_1} m_1$ が成り立つ．よって，$N = N_1 \parallel N_2$ の定義より，$I \xrightarrow{t_1}_{N} \cdots \xrightarrow{t_n}_{N} m_1 \uplus m_2$ かつ $\mathit{Enabled}_2(t_{n+1}, m_1 \uplus m_2)$ が成り立つ．補題の一番目の条件より，$\mathit{Enabled}_1(t_{n+1}, m_1 \uplus m_2)$ が成り立つので，$m_1 \xrightarrow{t_{n+1}}_{N_1}$ が成り立つ．ゆえに，$s = t_1 \cdots t_n t_{n+1} \in \mathbf{L}(N_1)$．

$s = t_1 \cdots t_n\downarrow$ の場合も同様．□

N_1 が決定性ペトリネットならば，上の条件は $\mathbf{L}(N_2) \subseteq \mathbf{L}(N_1)$ の必要条件でもある．

補題 4.10 $N_1 = (P_1, P_{a1}, T_1, I_1)$ を決定性ペトリネット，$N_2 = (P_2, P_{a2}, T_2, I_2)$ をペトリネットとし，$N_1 \parallel N_2 = (P, P_a, T, I)$ をそれらの合成ペトリネットとする．$\mathbf{L}(N_2) \subseteq \mathbf{L}(N_1)$ が成り立つならば，$N_1 \parallel N_2$ が到達可能な状態 m はすべて次の 2 つの条件を満たす．

(1) $\forall t \in \{!, ?\}.\mathit{Enabled}_2(t, m) \Rightarrow \mathit{Enabled}_1(t, m)$

(2) $\mathit{Accepting}_2(m) \Rightarrow \mathit{Accepting}_1(m)$

証明： $\mathbf{L}(N_2) \subseteq \mathbf{L}(N_1)$ であるとする．$m = m_1 \uplus m_2$，$I \xrightarrow{t_1}_{N} \cdots \xrightarrow{t_n}_{N} m$ かつ $\mathit{Enabled}_2(t, m)$ を仮定すると，$I_2 \xrightarrow{t_1}_{N_2} \cdots \xrightarrow{t_n}_{N_2} m_2 \xrightarrow{t}_{N_2}$ が成り立つ．よって，$t_1 \cdots t_n t \in \mathbf{L}(N_2) \subseteq \mathbf{L}(N_1)$ であり，ある m_1' について $I_1 \xrightarrow{t_1}_{N_1} \cdots \xrightarrow{t_n}_{N_1} m_1' \xrightarrow{t_{n+1}}_{N_1}$ が成り立つ．ところが，$I_1 \xrightarrow{t_1}_{N_1} \cdots \xrightarrow{t_n}_{N_1} m_1$ かつ N_1 は決定性ペトリネットなので，$m_1 = m_1'$．よって，$\mathit{Enabled}_1(t, m)$ が成り立つ．

条件 $\mathit{Accepting}_2(m) \Rightarrow \mathit{Accepting}_1(m)$ の証明も同様．□

以上の補題とペトリネットの到達可能性の決定可能性より，以下の定理が得られる．

定理 4.11 D, U を使用法表現とする．$N(D)$ が決定性ペトリネットならば，$D \leq U$ は決定可能である．

5. 実　　装

これまでの議論に基づき，並行プログラミング言語 Pict [13] の核部分に使用法表現つきの宣言を付加した言語 `PictUs` を設計し，型検査器を実装した．Pict の高階型などには現時点では対応していない．実装にあたっては，Pict の処理系の構文解析の部分は再利用し，使用法表現を考慮した型検査器は新たに Objective Caml で記述した．また，ペトリネットの到達可能性問題を整数線形計画問題に近似して解くために，Omega library [14] を用いている．

たとえば，

$(\nu x : [\mathbf{bool}]\ \mathbf{chan}_{(!\,|\,?)})\,(x![\mathbf{true}].\mathbf{0} \mid x?[y : \mathbf{bool}].\mathbf{0})$

は `PictUs` では以下のように記述できる．

```
new x:chan(!|?,Bool)
run (x![true] | x?[y:Bool] = ())
```

ここで，`!|?` が使用法表現の宣言であり，通信チャネル `x` を送信と受信に 1 回ずつ用いるべきであることを表す．

型検査の実行例を示す．以下のプロセスを考える．

$$
\begin{array}{l}
(\nu x : [\mathbf{bool}]\ \mathbf{chan}_{(!\,|\,?)}) \\
(\nu y : [[\mathbf{bool}]\ \mathbf{chan}_{!}]\ \mathbf{chan}_{(!\,|\,?)}) \\
(y![x].\mathbf{0} \mid (y?[z : [\mathbf{bool}]\ \mathbf{chan}_{!}].z![\mathbf{true}].\mathbf{0} \\
\qquad \mid x?[n : \mathbf{bool}].\mathbf{0}))
\end{array}
$$

上式を `PictUs` の構文で記述すると，以下のようになる．

```
new x:chan(!|?,[Bool])
new y:chan(!|?,[chan(!,[Bool])])
run (y![x] | y?[z:chan(!,[Bool])] =
      z![true] | x?[n:Bool] = ())
```

これを我々の型検査器に入力として与えると，以下のような出力が得られる．

```
Subusage Constraint
(!<!)
((!|?)<(!|?))
((!|?)<(!|?))
(!<!)

Typability:yes
```

2–5 行目が前節のステップ 2 で得られた部分使用法表現の制約であり，最後の行が型検査の結果である．この場合は yes なので，プロセスが型付けできる（すなわちすべての通信チャネルが正しく使用されている）ことを表す．

また，1 章の ping サーバの例は，PictUs では

```
new ping:
     chan(((*?)|(*!)),[chan(!,[String])])
run ping?[reply:chan(!,[String])]=
     reply![“ok”]
```

と表現できる．これを型検査すると

```
Subusage Constraint
(!<!)
((*(?)|*(!))<?)
(!<!)

Typability:yes
```

という結果となり，通信チャネルが正しく使用されていることが分かる．

一方，

```
new ping:
     chan(((*?)|(*!)),[chan(!,[String])])
run ping?[reply:chan(!,[String])]=
     (reply![“ok”]|reply![“ok”])
```

を型検査すると

```
Subusage Constraint
(!<(!|!))
((*(?)|*(!))<?)
(!<!)

Typability:no
```

となって，誤りを検出することができる．

注 5.1 Pict では，本論文で扱った言語と異なり，送信後の動作を記述することはできない．すなわち，$x![v_1,\ldots,v_n].P$ の P はつねに $\mathbf{0}$ である．したがって，注意深い読者は，$!.U$ の形の使用法表現が不要であり，部分使用法関係の判定が簡単化されると思うかもしれない．しかしながら，Pict では再帰型を許すため，$x![x, v_2\ldots,v_n].\mathbf{0}$ のようなプロセスにおける x の使用法を表すために，$!.U$ の形の使用法表現が必要となる．

6. 関 連 研 究

並行プログラミング言語において本研究の使用法表現のようなチャネルの使用法の宣言を許した型システムとしては，小林らによる線形型システム[10]，Chaki らによる研究[15] がある．小林らの型システムでは，宣言できる使用法が本論文での $(!\,|\,?)$ および $\mu\alpha.(0\&?.\alpha\&!.\alpha)$ に制限されており，部分使用法関係の判定は自明である．Chaki らの型システムでは，CCS のプロセスを型として宣言できるが，型判定問題は決定不能である．

また，小林ら[16] は π 計算にファイルやメモリなどの資源へのアクセスプリミティブを追加した言語における資源使用法解析のための型システムを提案，実装している．彼らの枠組みにおける計算資源の使用法の宣言が本研究での使用法表現におおよそ対応し，本論文と同様，型判定問題が宣言された計算資源の使用法と（CCS のプロセスとして表される）推論された使用法との間の包含関係の判定問題に帰着されている．ただし，彼らの枠組みでは計算資源の使用法として宣言できる言語は正則言語に制限されており，本論文の決定性ペトリネット言語という制限よりもきつい．また，宣言できる場所は計算資源の生成（本論文でのチャネル生成）個所のみである．さらに，以上の制限の下でも，彼らの型システムに対する型判定問題は決定可能ではなく，近似が必要である．

本研究では宣言できる使用法表現を決定性ペトリネット言語に制限したが，この制限をはずすと型判定問題が決定不能になることは，BPP のトレース包含判定問題の決定不能性から導かれる[6),7)]．Hirshfeld [6)] は，Minsky マシンの停止性問題を BPP のトレース包含判定問題に帰着することにより，8 種類以上のラベルを持つ BPP のプロセスのトレース包含判定問題の決定不能性を示した．本研究で扱う使用法表現はラベルの種類を 2 に制限した BPP に相当するが，小林と須藤[7] はそのような制限の下でもトレース包含判定問題が決定不能であることを示した．

7. 結論と今後の課題

本研究では，通信チャネルの使用法を明示的に宣言できる並行プログラミング言語の型システムを与え，その型検査アルゴリズムを示した．これによって，プログラマが通信チャネルの用途を宣言し，プログラムがその宣言に従っているかどうかを型検査によって静的に検証することが可能となった．

今後の課題として，まず使用法表現の拡張とそれにともなう部分使用法関係の定義の見直しがあげられる．デッドロックを解析するための小林の型システム[3]では，使用法の基本アクションに “capability” と “obligation” という属性が付加されており，それを用いて通信チャネルの使用法をよりきめ細かく表現することができる．そのような使用法表現の拡張をしたうえでも本論文と同様に部分使用法関係の判定が行えるかどうかは今後の課題である．

また，現在の `PictUs` の実装は Pict の高階関数（kind やチャネルを介した型引数の受け渡し）に未対応であり，Pict のフルセットや CML，Java などの実際の言語への応用も今後の課題である．

参 考 文 献

1) Sumii, E. and Kobayashi, N.: A Generalized Deadlock-Free Process Calculus, *Proc. Workshop on High-Level Concurrent Language* (*HLCL'98*), ENTCS, Vol.16(3), No.3, pp.55–77 (1998).
2) Kobayashi, N., Saito, S. and Sumii, E.: An Implicitly-Typed Deadlock-Free Process Calculus, *Proc. CONCUR2000*, Lecture Notes in Computer Science, Vol.1877, pp.489–503, Springer-Verlag (2000).
3) Kobayashi, N.: A New Type System for Deadlock-Free Processes, *Proc.CONCUR 2006*, Lecture Notes in Computer Science, Vol.4137, pp.233–247, Springer-Verlag (2006).
4) Kobayashi, N.: Type-Based Information Flow Analysis for the Pi-Calculus, *Acta Informatica*, Vol.42, No.4-5, pp.291–347 (2005).
5) Christensen, S., Hirshfeld, Y. and Moller, F.: Decomposability, Decidability and Axiomatisability for Bisimulation Equivalence on Basic Parallel Processes, *LICS*, pp.386–396 (1993).
6) Hirshfeld, Y.: Petri Nets and the Equivalence Problem, *CSL*, pp.165–174 (1993).
7) Kobayashi, N. and Suto, T.: Undecidability of 2-Label BPP Equivalences, in preparation (2007).
8) Milner, R.: *Communicating and Mobile Systems: the Pi-Calculus*, Cambridge University Press (1999).
9) Sangiorgi, D. and Walker, D.: *The Pi-Calculus: A Theory of Mobile Processes*, Cambridge University Press (2001).
10) Kobayashi, N., Pierce, B.C. and Turner, D.N.: Linearity and the Pi-Calculus, *ACM Trans. Prog. Lang. Syst.*, Vol.21, No.5, pp.914–947 (1999).
11) Peterson, J.L.: *Petri Net Theory and the Modeling of Systems*, Prentice-Hall (1981).
12) Pelz, E.: Closure Properties of Deterministic Petri Nets, *STACS 87: 4th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, Vol.247, pp.371–382, Springer-Verlag (1987).
13) Pierce, B.C. and Turner, D.N.: Pict: A Programming Language Based on the Pi-Calculus, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Plotkin, G., Stirling, C. and Tofte, M. (Eds.), pp.455–494, MIT Press (2000).
14) The Omega Library Version 1.1.0 Interface Guide. http://www.cs.umd.edu/projects/omega
15) Chaki, S., Rajamani, S. and Rehof, J.: Types as Models: Model Checking Message-Passing Programs, *Proc. ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pp.45–57 (2002).
16) Kobayashi, N., Suenaga, K. and Wischik, L.: Resource Usage Analysis for the Pi-Calculus, *Logical Methods in Computer Science*, Vol.2, No.3:4, pp.1–42 (2006).

付 録

A.1 4 章で用いた補題

補題 A.1.1 U を自由変数として α のみを含む使用法表現，U' を閉じた使用法表現とすると，$U' \leq [U'/\alpha]U$ ならば $U' \leq \mu\alpha.U$ が成り立つ．

証明（概略）： $U' \leq [U'/\alpha]U$ かつ $s \in [\![\mu\alpha.U]\!]$ が成り立つと仮定する．このとき，$s \in [\![U']\!]$ であることを示せばよい．U に代入 $[U/\alpha]$ を n 回適用して得られる使用法表現を $U_n = ([U/\alpha])^n U$ とすると，図 1 の規則より，ある n について $s \in [\![[0/\alpha]U_n]\!]$ が成り立つ．これより，$s \in [\![[U'/\alpha]U_n]\!]$ が成り立つ．ここで，$U' \leq [U'/\alpha]U$ より，

$$\begin{aligned}[U'/\alpha]([U/\alpha])^n U &= [[U'/\alpha]U/\alpha]([U/\alpha])^{n-1}U \\ &\geq [U'/\alpha]([U/\alpha])^{n-1}U \\ &\geq \cdots \geq U'\end{aligned}$$

が成り立つ．よって，$s \in [\![U']\!]$ が得られる．□

補題 A.1.2 部分使用法関係の集合 C が左辺に α を含まないとき，$\{\alpha \leq U\} \cup C$ が充足可能であることと $[\mu\alpha.U/\alpha]C$ が充足可能であることは等価である．

証明：

- “Only if”: 代入 θ が $\{\alpha \leq U\} \cup C$ の解である，すなわち $\{\theta\alpha \leq \theta U\} \cup \theta C$ が成り立つと仮定する．このとき，$\theta_1 = \theta \backslash \alpha$ が $[\mu\alpha.U/\alpha]C$ の解であることを示す．今，$\theta\alpha = U_\alpha$ とすると，仮定より $U_\alpha \leq [U_\alpha/\alpha](\theta_1 U)$ が成り立つ．ここで，補題 A.1.1 を用いると，$U_\alpha \leq \mu\alpha.(\theta_1 U)$ が得られる．よって，C の任意の要素 $U_1 \leq U_2$ に対して，
$$\begin{aligned}\theta_1[\mu\alpha.U/\alpha]U_1 &= \theta_1 U_1 \\ &= \theta U_1 \\ &\leq \theta U_2 \\ &= [U_\alpha/\alpha]\theta_1 U_2 \\ &\leq [\mu\alpha.(\theta_1 U)/\alpha]\theta_1 U_2 \\ &= \theta_1[\mu\alpha.U/\alpha]U_2\end{aligned}$$
が成り立つ．ゆえに，θ_1 は $[\mu\alpha.U/\alpha]C$ の解である．
- “If”: 代入 θ' が $[\mu\alpha.U/\alpha]C$ の解とすると，$\theta = [\mu\alpha.U/\alpha] \circ \theta'$ が $\{\alpha \leq U\} \cup C$ の解であることより明らか．

□

(平成 18 年 12 月 15 日受付)
(平成 19 年 3 月 13 日採録)

須藤 崇

1983 年生．東北大学大学院情報科学研究科情報基礎科学専攻．

小林 直樹（正会員）

1968 年生．1991 年東京大学理学部情報科学科卒業．1993 年同大学大学院理学系研究科情報科学専攻修士課程修了，同年博士課程進学．東京大学大学院理学系研究科情報科学専攻助手，講師，東京工業大学大学院情報理工学研究科助教授を経て 2004 年より東北大学大学院情報科学研究科教授，現在に至る．博士（理学）．型理論，プログラム解析，並行計算等に興味を持つ．ACM 会員．2003 年日本 IBM 科学賞受賞．