情報処理学会論文誌　プログラミング　Vol. 1　No. 2　71–84 (Sep. 2008)

# 型エラースライシングによるデッドロックの原因特定

飯村枝里[†1]　小林直樹[†1]　末永幸平[†2]

並行プログラムは，実行の非決定性などのために，バグが混入しやすく，またそのバグの発見が難しい．そこで，並行プログラムを静的かつ網羅的に検証するための手法として，型システムを用いた手法が近年注目されている．小林らは $\pi$ 計算を対象としてデッドロックの検証を行う型システムを提案し，それに基づく自動検証器を実装している．しかしながら，型システムとそれに基づく検証アルゴリズムの複雑さのため，検証に失敗した場合に具体的にプログラムのどの部分に誤りがあるのかをユーザが判断するのは困難であった．本研究では，関数型言語を対象として提案されている型エラースライシングの考え方を，小林らの型システムに適用することにより，デッドロックの可能性がある場合にその原因個所を具体的に提示するアルゴリズムを与える．さらに小林の検証器に提案手法の実装を組み込み，その有効性を確認した．

## Identifying Deadlock Errors by Type Error Slicing

Eri Iimura,[†1] Naoki Kobayashi[†1] and Kohei Suenaga[†2]

It is difficult to find bugs in concurrent programs because of non-determinism. Recently, various type systems have been proposed for detecting bugs in concurrent programs. For instance, Kobayashi et al. present type systems for checking deadlock freedom, livelock freedom, and other properties of the $\pi$-calculus. Their type systems are implemented in the tool TyPiCal. However, it is sometimes difficult for the users to understand the reason for a type error reported by TyPiCal. In this paper, drawing on research in type error slicing for functional languages, we propose a slicing algorithm for identifying deadlock type errors reported by TyPiCal. We have implemented our slicing algorithm as an extension to TyPiCal, and verified its correctness on a small number of examples.

†1 東北大学大学院情報科学研究科
Graduate School of Information Sciences, TOHOKU University
†2 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

## 1. はじめに

並行プログラムは，動作の非決定性などのために，プログラムの記述が難しく，テスト実行などによるデバッグが難しい．そこで，並行プログラムの様々な性質を静的に効率良く検証するための型システムがこれまでに数多く提案されている[1)–3),6),7),9)–12),15)]．しかしながら，それらの多くでは，型システムの複雑さのために，検証に失敗して型エラーと判定された場合に，ユーザによる誤りの原因部分の特定が難しい．たとえば，Kobayashi によるデッドロックやライブロックなどの検証器 `TyPiCal` [9)] は，どの送受信が必ず成功するかを解析して表示するが，成功しない送受信についてその原因を特定できる情報を出力することができない．

上記の問題を解決するため，本研究では小林によるデッドロック解析のための型システム[9),12)] を対象とし，デッドロックの原因個所を特定するための型エラースライシングの手法を提案する．型エラースライシングとは，関数型言語を対象として Haack ら[5),8)] によって導入されたものであり，プログラムに型エラーが存在する場合に，型エラーの原因となっている部分のみをプログラムスライスとして抽出する手法である．表示されたスライスから，プログラムのどの部分でどのような型エラーが起きているのかを理解するのが容易になる．

デッドロック解析のための型エラースライスの例を以下にあげる．次のような $\pi$ 計算のプロセスを考える．

$$s?b.\,(\text{if } b \text{ then } r_1?x.\,r_2!x \text{ else } r_2!0)$$
$$\mid s!\text{true}.\,r_2??y.\,r_1!y$$

このプロセスは 2 つのプロセスを並行に実行する（縦棒が並行実行を表す）．1 つ目のプロセスは，まず通信チャネル $s$ から受信を行う．受信した値 $b$ が真であれば通信チャネル $r_1$ から受信を行い，受け取った値 $x$ を $r_2$ に送信する．値 $b$ が偽であった場合には，$r_2$ に整数 0 を送信する．2 つ目のプロセスは，チャネル $s$ に true を送信した後，チャネル $r_2$ から受信を行い，受け取った値を $r_1$ に送信する．ここで，2 つ目のプロセスの $r_2$ からの受信が ? ではなく ?? となっているのは，この受信が必ず成功すべきであるというプログラマの意図を表している．本論文で扱うデッドロックとは，そのような必ず成功すべきと宣言された送受信がブロックされたままプログラム全体が停止してしまうような状態を指す．デッドロック解析の目的はそのようなデッドロックが起きないことを静的に検証することである．

上記のプログラムでは，1 つ目のプロセスの then 部と 2 つ目のプロセスとで $r_1$，$r_2$ に関する送受信の順序が食い違っているため，デッドロックが起きる（すなわち，$r_2??y.\,r_1?y$

が成功しないままプログラム全体が止まってしまう）．このプログラムに対し，本研究の型エラースライシングを適用すると，以下のようなスライスが得られる．

$\cdots$(if $\cdots$ then $r_1?x.\, r_2!x$ else $\cdots$)
| $\cdots r_2??y.\, r_1!y$

これにより，1 つ目のプロセスの then 部と 2 つ目のプロセスにおける $r_1$，$r_2$ を介した送受信に問題があることが分かる．

Haack らによる通常の型エラースライシング[5),8)] は，次のようなステップからなる．

(1)　プログラムが型付けできるための必要十分条件を制約の集合として取り出す．その際，どの制約がどのプログラムの文面に起因するのかが分かるように各制約にラベル付けをしておく．
(2)　(1) で抽出した制約集合が充足可能であれば，型付けできるとして終了．充足不能であれば，充足不能な極小部分集合を求める．
(3)　(2) で求まった極小部分集合に含まれる制約の元となるプログラムの文面をスライスとして抽出する．削除されて「$\cdots$」で置き換えられた部分を任意の型を持ちうる「万能な項」と見なせば，抽出されたプログラムスライスは，型付けできないような極小のプログラムスライスとなる．

しかしながら，上の型エラースライシングの考え方を小林のデッドロックの型システム[12)] にそのまま適用することは，以下の理由から自明ではない．

- まず第 1 に，どの制約をどのプログラムの文面と対応付けるべきかが自明でない．たとえば，小林の型システム[12)] では，受信のための規則は以下のようになっている．

$$\frac{\Gamma, y:\tau \vdash P}{x:\mathrm{ch}(\tau, ?^{t_o}_{t_c});\Gamma \vdash x?y.\, P}$$

型環境に関する演算 $x:\mathrm{ch}(\tau, ?^{t_o}_{t_c});\Gamma$ の意味はここでは説明しないが，重要なのは，$x$ に関する受信の存在が，型環境中の $x$ 以外の変数の型にも影響を及ぼすという点である．このため，小林の型システム[12)] に対して単純に型エラースライシングを施すと，ある送受信がデッドロックに関係あると判定された場合，その前に行われるすべての送受信がスライスとして残ってしまう．たとえば，上のプログラム例では，$s$ に関する送受信もスライスとして残ってしまう（詳細は 4.1 節の議論を参照されたい）．
- 第 2 に，より本質的な問題として，スライスによって削除された部分を一律に「万能なプロセス」と解釈すると，型付けできないような極小のスライスが必ずしもデッドロックの原因を表すものとならない．上のプログラム例の理想とするスライス

$\cdots$(if $\cdots$ then $r_1?x.\, r_2!x$ else $\cdots$)
| $\cdots r_2??y.\, r_1!y$

において，最初の $\cdots$ を任意の送受信ができるものと解釈すると，($r_2$ への送信も行えることになるので）型付けできることになってしまう．したがって，この場合の $\cdots$ は「$r_1$，$r_2$ 以外のチャネルについてのみ」万能な動作と見なすべきである．一方，上のプログラム例に対する別のスライスとして

$\cdots$(if $\cdots$ then $r_1?x.\, r_2!x$ else $\cdots$)
| $\cdots r_2??y.\, \cdots$

を考えると，$r_2??x$ の後の $\cdots$ がどのような万能なプロセスであっても，$r_2$ からの受信が成功しないかぎり $r_1$ への送信が行えないため，正しい型エラースライスと判定されてしまう．しかしながら，このスライスからは $r_1$ と $r_2$ に関する送受信の依存関係の循環のためにデッドロックするという情報は得られない．

上記の問題（本来スライスとして残したい情報が削除されてしまったり，逆に削除したい情報がスライスとして残ってしまうという問題）を解決するため，小林の型システム[12)] と推論アルゴリズム自身に改良を加える．1 番目の問題は，送受信の型付け規則に付随する制約の形を変更するとともに，制約のラベル付けに工夫を施すことによって解決する．2 番目の（後者の）問題に対しては，スライシングによって削除されたプロセスは，ただの「万能なプロセス」ではなく，直前の動作によってブロックされない「超越的なプロセス」と解釈されるように，型の概念を変更する．たとえば，上の例では，間違ったスライス

$\cdots$(if $\cdots$ then $r_1?x.\, r_2!x$ else $\cdots$)
| $\cdots r_2??y.\, \cdots$

における最後の「$\cdots$」は超越的なプロセスなので $r_1$ への送信をいつでも行えるものと（型判定の際に）解釈される．その結果，スライス全体が型付けできる（したがってデッドロックしない）ことになり，型エラースライスとは見なされない．これらの詳細については 3 章および 4 章を参照されたい．

以降の論文の構成は次のとおり．まず 2 章で本研究の対象言語を与え，デッドロックを定義する．3 章では型エラースライシングを行う基礎となる，デッドロックの解析のための型システムを定義する．4 章で型エラースライシングのアルゴリズムを与え，5 章で実装と実験結果について述べる．6 章で関連研究について議論し，7 章で結論を述べる．

## 2. 対 象 言 語

本研究における対象言語の構文を以下に示す．$\pi$ 計算にブール値と組を追加したものであり，各式にはプログラム中の位置を表すラベルを付加している．なおラベルが重要でない場合には今後これを省略する．

$$
\begin{aligned}
P \quad &::= \mathbf{0}^l \mid (P \mid Q) \mid *P \mid (\nu x)^l\, P \\
&\quad \mid x!^l v.\, P \mid x!!^l v.\, P \mid x?^l y.\, P \mid x??^l y.\, P \\
&\quad \mid (\text{if } v \text{ then } P \text{ else } Q)^l \mid \text{let } x = e \text{ in } P \\
e \quad &::= \text{true} \mid \text{false} \mid x^l \mid \langle e_1, e_2 \rangle \\
&\quad \mid \text{proj}_1^l(e) \mid \text{proj}_2^l(e) \\
v \quad &::= \text{true} \mid \text{false} \mid x^l \mid \langle v_1, v_2 \rangle
\end{aligned}
$$

構文中の $l$ はラベルを表す．0 は何もしないプロセスである．$x!^l v.\, P$ はチャネル $x$ を介し値 $v$ を送信し，($v$ が受信された後に）$P$ に従って通信を行うプロセスである．$x?^l y.\, P$ はチャネル $x$ を介して値を受信し，$y$ を受信した値に束縛した後 $P$ を実行するプロセスである．プロセス $x!!^l v.\, P$ と $x??^l y.\, P$ は，操作的にはそれぞれ $x!^l v.\, P$，$x?^l y.\, P$ と同じであるが，その送受信が成功すべきであるというユーザの意図を表す．以下では，$x!!^l v.\, P$ と $x??^l y.\, P$ を**マーク付きプロセス**と呼ぶ．プロセス $P \mid Q$ は $P$ と $Q$ を並行に実行することを表す．プロセス $*P$ は $P$ を無限個複製し，それらを並行に実行するプロセスである．$(\nu x)^l\, P$ は新しい通信チャネル $x$ を生成し，$P$ を実行するプロセスである．プロセス $(\text{if } v \text{ then } P \text{ else } Q)^l$ は $v$ が true であれば $P$ を，false であれば $Q$ を実行する．プロセス $\text{let } x = e \text{ in } P$ は $x$ を $e$ に束縛し $P$ を実行する．以降しばしば末尾の 0 を省略し，$x!v.\, 0$，$x?y.\, 0$ をそれぞれ $x!v$，$x?y$ と書く．

**図 1** に操作的意味を示す．図中，$P \equiv Q$ は $(P \preceq Q) \wedge (Q \preceq P)$ を意味する．関係 $e \Downarrow v$ は式 $e$ の評価結果が $v$ であることを表す．関係 $P \preceq Q$ は，プロセス $P$ が通信をともなわずに $Q$ に遷移できることを表す．関係 $P \to Q$ は，プロセス $P$ が通信を行って $Q$ に遷移できることを表す．また，$P \to Q$ なる $Q$ が存在しないとき，$P \not\to$ と書く．

**定義 2.1（デッドロック）** プロセス $P$ が**デッドロックに陥っている**とは，$P$ のトップレベルにマーク付きプロセスが存在する（すなわち $P \preceq (\widetilde{\nu u})(x!!v.\, P_1 \mid P_2)$ または $P \preceq (\widetilde{\nu u})(x??y.\, P_1 \mid P_2)$ が成り立つ）にもかかわらず，$P$ が簡約できないことを表す．プロセス $P$ がどのような簡約列によってもデッドロックに陥らないとき，$P$ は**デッドロックフリー**であるという．

(値の評価規則)

$$
x \Downarrow x \qquad \frac{b \in \{\text{true}, \text{false}\}}{b \Downarrow b} \qquad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{\langle e_1, e_2 \rangle \Downarrow \langle v_1, v_2 \rangle} \qquad \frac{e \Downarrow \langle v_1, v_2 \rangle \quad i \in \{1, 2\}}{\text{proj}_i(e) \Downarrow v_i}
$$

(プロセスの構造的前順序関係)

$$
P \equiv P \mid 0 \qquad *P \preceq *P \mid P \qquad \text{if true then } P \text{ else } Q \preceq P
$$

$$
P \mid Q \equiv Q \mid P \qquad \frac{P \preceq P'}{P \mid Q \preceq P' \mid Q} \qquad \text{if false then } P \text{ else } Q \preceq Q
$$

$$
P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad \frac{e \Downarrow v}{\text{let } x = e \text{ in } P \preceq [v/x]P}
$$

$$
\frac{x \text{ not free in } Q}{(\nu x)\, P \mid Q \equiv (\nu x)\, (P \mid Q)} \qquad \frac{P \preceq Q}{(\nu x)\, P \preceq (\nu x)\, Q}
$$

(プロセスの遷移関係)

$$
x!v.\, P \mid x?y.\, Q \to P \mid [v/y]Q \qquad \frac{P \to Q}{(\nu x)\, P \to (\nu x)\, Q}
$$

$$
\frac{P \to Q}{P \mid R \to Q \mid R} \qquad \frac{P \preceq P' \quad P' \to Q' \quad Q \preceq Q'}{P \to Q}
$$

**図 1**　対象言語の操作的意味
Fig. 1　Operational semantics.

**例 2.1**　次のプロセス $P$ はデッドロックに陥っている．

$r_1?x.\, r_2!x \mid r_2??y.\, r_1!y$

一方，次のプロセス $P'$ は受信にマークが付いていないのでデッドロックではない．

$r_1?x.\, r_2!x \mid r_2?y.\, r_1!y$

1 章であげたプロセス

$s?b.\, (\text{if } b \text{ then } r_1?x.\, r_2!x \text{ else } r_2!0)$

$\mid s!\text{true}.\, r_2??y.\, r_1!y$

は上記のプロセス $P$ に簡約されるのでデッドロックフリーではない．一方，次のプロセスはデッドロックフリーである．

$s?b.(\text{if } b \text{ then } r_1?x.r_2!x \text{ else } r_2!0)$
$\mid s!\text{true}.r_1!0.r_2??y$

## 3. 型システム

本章では，型エラースライシングの元となるデッドロック解析のための型システムとして，小林の型システム[12] に若干の変更を加えたものを導入する．本章の型システムは，義務レベルとして $-\infty$ が加わっている以外は小林の型システムと同じである．したがって，以下では型システムの要点についてのみ説明し，より詳細な説明は関連論文[12),13)] に譲る．義務レベルとして $-\infty$ を加えたのは，1 章で触れた「超越的なプロセス」の能力を表現するためであり，1 章で述べた型エラースライシングを行ううえでのその他の問題点については，次章で述べる制約生成の段階で解決する．

### 3.1 使用法表現と型

型の構文およびそれに関する演算や関係を**図 2** にまとめた．デッドロック解析のための型システムのアイデアは，通信チャネルの型に，各チャネルがどのような順序で送受信に使われるかを表す**使用法表現**を付加することである．図 2 中の $\tau$ が型を表し，$\mathrm{ch}(\tau, U)$ は型 $\tau$ の値を使用法表現 $U$ に従って送受信するためのチャネルの型を表す．たとえば，$\mathrm{ch}(\mathrm{bool}, !|?)$ は，ブール値の送信と受信に 1 回ずつ用いるためのチャネルの型を表す．使用法表現中の各送信（と受信）に対応して受信（と送信）が存在することを検査することにより，各通信チャネルが正しく使われていることを確認できる．また，$r_1?x.r_2!x \,|\, r_2??y.r_1!y$ のように，複数のチャネルに関する送受信間の循環的な依存関係を排除するため，使用法表現中の各送受信（図 2 中の $\alpha$）には，**義務レベル**と**権限レベル**が付加されている．これらは $\mathbf{Nat} \cup \{\infty, -\infty\}$ の要素であり，直感的には，義務レベルが送受信を行わなければならない義務の度合いを表し，権限レベルが，送受信を行う権限の度合いを表す．循環的な依存関係を防ぐため，レベル $n$ の義務は，レベル $n$ 未満の権限のみを用いて果たさなければならない．たとえば，$r_1?x.r_2!x$ において，$r_2$ を介した送信は $r_1$ を介した受信が成功する場合にのみ実行されるため，前者の義務レベルよりも後者の権限レベルの方が小さくなければならない．義務レベルが $\infty$ の場合には，送受信の義務がないことを，権限レベルが $\infty$ の場合には，送受信を行った場合にそれが成功する保証がない（デッドロックするかもしれない）ことを表す．義務レベルの $-\infty$ は本論文で新たに導入されたものである．通常のプロセスでは，レベル $-\infty$ の義務を果たすことはできず，型エラースライスによって導入される超越的なプロセス「$\cdots$」のみによって果たすことができる．

図 2 中の各使用法表現の直感的な意味は以下のとおりである．0 はそのチャネルがまったく使われないことを表す．$\alpha^{t_1}_{t_2}.U$ はチャネルが $\alpha$ の表す動作に従って使用された後に $U$ に従った使われ方をすることを表す．$\alpha$ は ? または ! であり，? はチャネルが受信に 1 回使われることを表し，! は送信に 1 回使われることを表す．また，$t_1$ が義務レベル，$t_2$ が権限レベルを表す．

$U_1 \,|\, U_2$ は $U_1$ と $U_2$ に従って並行に用いられることを表す．$\uparrow^t U$ は，$U$ の義務レベルを $t$ まで引き上げることを表す．$U_1 \,\&\, U_2$ は $U_1$ もしくは $U_2$ のどちらかに従ってチャネルが使われることを表す．$\rho$ は再帰に用いられる変数（以下，使用法変数と呼ぶ）である．$\mu\rho.U$ は $\rho = U$ を満たす $\rho$ に従って再帰的に用いられることを表す．たとえば $\mu\rho.?.\rho$ は無限回受信に使われることを表す．$*U$ は，$U$ に従って無限回並行に使用されることを表す．ここで $\mu\rho.U$ 中の $\rho$ は束縛変数であり，$U$ 中の自由な使用法変数の集合を $FV(U)$ と記す．また，$U_2$ 中の自由な使用法変数 $\rho$ を $U_1$ で置き換えた使用法表現を $[U_1/\rho]U_2$ と記す．なお，今後末尾の 0 はしばしば省略し，たとえば !.0 や ?.0 は ! や ? と記述する．

図 2 中の $ob_\alpha(U)$ と $cap_\alpha(U)$ はそれぞれ使用法表現 $U$ の義務レベルと権限レベルを表す．なお，$\overline{\alpha}$ は $\alpha$ と反対の動作を表す（$\overline{?} = !$, $\overline{!} = ?$）．特に，$cap_\alpha(\mu\rho.\rho) = \infty$，$ob_\alpha(\mu\rho.\rho) = -\infty$ である（一方，小林の型システム[12] では $cap_\alpha(\mu\rho.\rho) = \infty$, $ob_\alpha(\mu\rho.\rho) = 0$ である）．また $ob_\alpha(\uparrow^t U)$ は $ob_\alpha(U)$ が $-\infty$ のときに，$t$ の値にかかわらず $-\infty$ となることに注意されたい．

使用法表現中の送受信の対応がとれていることを表す述語 $rel(U)$ を以下のように定義する．

**定義 3.1（reliability）** $ob_{\overline{\alpha}}(U) \leq cap_\alpha(U)$ が成り立つとき $con_\alpha(U)$ と書き，$con_!(U) \wedge con_?(U)$ が成り立つとき $con(U)$ と書く．$U \to^* U'$ なる任意の $U'$ について $con(U')$ のとき $rel(U)$ と書く．

図 2 中の部分型関係 $\tau_1 \leq \tau_2$ の定義で用いている**部分使用法関係** $U_1 \leq U_2$ は，$U_1$ が $U_2$ よりも制限の緩い使用法であること，すなわち，$U_1$ に従って用いられるべきチャネルは $U_2$ に従って用いられてもよいことを表す．正確な定義は複雑なので関連論文[12] に譲り，以下では部分使用法関係が満たす重要な性質だけ列挙する．

**補題 3.1（部分使用法関係の性質）**

(1)　$U \leq U_1 \wedge U \leq U_2$ と $U \leq U_1 \,\&\, U_2$ は同値．
(2)　$\mu\alpha.U \leq [\mu\alpha.U/\alpha]U$．
(3)　$U_1 \leq [U_1/\alpha]U$ ならば $U_1 \leq \mu\alpha.U$．

Syntax of types

$$U ::= 0 \mid \alpha^{t_1}_{t_2}.U \mid (U_1 \mid U_2) \mid \uparrow^t U \mid U_1 \ \& \ U_2 \mid \rho \mid \mu\rho.U \mid *U$$
$$\alpha ::= ? \mid ! \qquad t \in \mathbf{Nat} \cup \{\infty, -\infty\}$$
$$\tau ::= \mathrm{bool} \mid \tau_1 \times \tau_2 \mid \mathrm{ch}(\tau, U)$$

Operational semantics of usages

$$U_1 \mid U_2 \preceq U_2 \mid U_1 \qquad (U_1 \mid U_2) \mid U_3 \preceq U_1 \mid (U_2 \mid U_3) \qquad *U \preceq *U \mid U$$
$$U_1 \,\&\, U_2 \preceq U_i \ (i \in \{1,2\}) \qquad \uparrow^t \alpha^{t_1}_{t_2}.U \preceq \alpha^{\mathbf{max}(t,t_1)}_{t_2}.U \qquad \uparrow^t (U_1 \mid U_2) \preceq (\uparrow^t U_1) \mid (\uparrow^t U_2)$$
$$\mu\rho.U \preceq [\rho \mapsto \mu\rho.U]U \qquad \frac{U_1 \preceq U_1' \quad U_2 \preceq U_2'}{U_1 \mid U_2 \preceq U_1' \mid U_2'} \qquad \frac{U \preceq U'}{\uparrow^t U \preceq \uparrow^t U'}$$
$$?.U_1 \mid !.U_2 \to U_1 \mid U_2 \qquad \frac{U_1 \to U_1'}{U_1 \mid U_2 \to U_1' \mid U_2} \qquad \frac{U_1' \to U_2' \quad U_1 \preceq U_1' \quad U_2 \preceq U_2'}{U_1 \to U_2}$$

$cap_\alpha(U), cap_\alpha(\tau), ob_\alpha(U), ob_\alpha(\tau)$

$$cap_\alpha(0) = cap_\alpha(\overline{\alpha}^{t_o}_{t_c}.U) = cap_\alpha(\rho) = \infty \qquad cap_\alpha(\alpha^{t_o}_{t_c}.U) = t_c$$
$$cap_\alpha(*U) = cap_\alpha(\mu\rho.U) = cap_\alpha(U)$$
$$cap_\alpha(\uparrow^t U) = cap_\alpha(U) \qquad cap_\alpha(U_1 \mid U_2) = \mathbf{min}(cap_\alpha(U_1), cap_\alpha(U_2))$$
$$cap_\alpha(U_1 \ \& \ U_2) = \mathbf{min}(cap_\alpha(U_1), cap_\alpha(U_2))$$
$$ob_\alpha(0) = ob_\alpha(\overline{\alpha}^{t_o}_{t_c}) = \infty \qquad ob_\alpha(\rho) = -\infty \qquad ob_\alpha(\alpha^{t_o}_{t_c}.U) = t_o$$
$$ob_\alpha(U_1 \mid U_2) = \mathbf{min}(ob_\alpha(U_1), ob_\alpha(U_2))$$
$$ob_\alpha(U_1 \ \& \ U_2) = \mathbf{max}(ob_\alpha(U_1), ob_\alpha(U_2))$$
$$ob_\alpha(*U) = ob_\alpha(\mu\rho.U) = ob_\alpha(U) \qquad ob_\alpha(\uparrow^t U) = \begin{cases} -\infty & \text{if } ob_\alpha(U) = -\infty \\ \mathbf{max}(t, ob_\alpha(U)) & \text{otherwise} \end{cases}$$
$$ob(\mathrm{bool}) = \infty \qquad ob(\tau_1 \times \tau_2) = \mathbf{min}(ob(\tau_1), ob(\tau_2)) \qquad ob(\mathrm{ch}(\tau, U)) = ob(U)$$

$\tau_1 \le \tau_2, \Gamma_1 \le \Gamma_2$

$$\mathrm{bool} \le \mathrm{bool} \qquad \frac{\tau_1 \le \tau_1' \quad \tau_2 \le \tau_2'}{\tau_1 \times \tau_2 \le \tau_1' \times \tau_2'} \qquad \frac{U \le U' \quad \tau \le \tau' \quad \tau' \le \tau}{\mathrm{ch}(\tau, U) \le \mathrm{ch}(\tau', U')}$$
$$\Gamma_1 \le \Gamma_2 \overset{\text{DEF}}{\Leftrightarrow} dom(\Gamma_1) \supseteq dom(\Gamma_2) \wedge \forall x \in dom(\Gamma_2).(\Gamma_1(x) \le \Gamma_2(x))$$
$$\wedge \quad \forall x \in dom(\Gamma_1) \setminus dom(\Gamma_2).(ob_?(\Gamma_1(x)) = ob_!(\Gamma_1(x)) = \infty)$$

Operations on types and type environments

$$*\mathrm{bool} = \mathrm{bool} \qquad *(\tau_1 \times \tau_2) = (*\tau_1) \times (*\tau_2) \qquad *\mathrm{ch}(\tau, U) = \mathrm{ch}(\tau, *U)$$
$$\uparrow^t \mathrm{bool} = \mathrm{bool} \qquad \uparrow^t (\tau_1 \times \tau_2) = (\uparrow^t \tau_1) \times (\uparrow^t \tau_2) \qquad \uparrow^t \mathrm{ch}(\tau, U) = \mathrm{ch}(\tau, \uparrow^t U)$$
$$\mathrm{bool} \mid \mathrm{bool} = \mathrm{bool} \qquad (\tau_1 \times \tau_2) \mid (\tau_1' \times \tau_2') = (\tau_1 \mid \tau_1') \times (\tau_2 \mid \tau_2')$$
$$\mathrm{ch}(\tau, U) \mid \mathrm{ch}(\tau, U') = \mathrm{ch}(\tau, (U \mid U'))$$
$$(*\Gamma)(x) = *(\Gamma(x))$$
$$(\Gamma_1 \mid \Gamma_2)(x) = \begin{cases} \Gamma_1(x) \mid \Gamma_2(x) \text{if } x \in dom(\Gamma_1) \cap dom(\Gamma_2) \\ \Gamma_1(x) \quad \text{if } x \in dom(\Gamma_1) \setminus dom(\Gamma_2) \\ \Gamma_2(x) \quad \text{if } x \in dom(\Gamma_2) \setminus dom(\Gamma_1) \end{cases}$$
$$(y : \mathrm{ch}(\tau, \alpha^{t_o}_{t_c}); \Gamma)(x) = \begin{cases} \mathrm{ch}(\tau, \alpha^{t_o}_{t_c}.U) \\ \qquad \text{if } x = y \wedge \Gamma(x) = \mathrm{ch}(\tau, U) \\ \mathrm{ch}(\tau, \alpha^{t_o}_{t_c}.0) \quad \text{if } x = y \wedge x \notin dom(\Gamma) \\ \uparrow^{t_c+1} \Gamma(x) \qquad \text{if } x \neq y \end{cases}$$

**図 2**　型と型に関する演算子および関係の定義
Fig. 2　Definition of types and their operations.

### 3.2 型　判　定

型判定には以下の 2 式を用いる．

$\Gamma \vdash e : \tau$

$\Gamma \vdash P$

前者は $e$ が型環境 $\Gamma$ の下で型 $\tau$ を持つことを意味し，後者は $P$ が型環境 $\Gamma$ の下で正しいプロセスであること，つまり $\Gamma$ に記されたチャネルの使用法どおりに通信を行うことを意味する．型環境 $\Gamma$ は変数から型への写像であり，$\emptyset$ は空の型環境を表す．また，型環境 $\Gamma$ の定義域を $dom(\Gamma)$ と書き，$\Gamma$ から $x$ を除いた型環境を $\Gamma \setminus x$ と書く．

$$\frac{\tau' \le \tau}{x : \tau' \vdash x : \tau}\ (\textsc{Tv-Var}) \qquad \frac{b \in \{\mathrm{true}, \mathrm{false}\}}{\emptyset \vdash b : \mathrm{bool}}\ (\textsc{Tv-Bool})$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \quad \Gamma_2 \vdash e_2 : \tau_2}{\Gamma_1 \mid \Gamma_2 \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}\ (\textsc{Tv-Pair}) \qquad \frac{\Gamma \vdash e : \tau_1' \times \tau_2' \quad \tau_i' \le \tau_i \quad ob(\tau_{3-i}) = \infty}{\Gamma \vdash \mathrm{proj}_i(e) : \tau_i}\ (\textsc{Tv-Proj})$$

$$\emptyset \vdash 0\ (\textsc{T-Zero}) \qquad \frac{\Gamma_1 \vdash P_1 \quad \Gamma_2 \vdash P_2}{\Gamma_1 \mid \Gamma_2 \vdash P_1 \mid P_2}\ (\textsc{T-Par}) \qquad \frac{\Gamma \vdash P}{*\Gamma \vdash *P}\ (\textsc{T-Rep})$$

$$\frac{\Gamma_1 \vdash P \quad \Gamma_2 \vdash e : \tau' \quad \tau' \le \tau}{x : \mathrm{ch}(\tau, !^0_{t_c}); (\Gamma_1 \mid \Gamma_2) \vdash x!e.\, P}\ (\textsc{T-Out}) \qquad \frac{\Gamma_1 \vdash P \quad \Gamma_2 \vdash e : \tau' \quad \tau' \le \tau \quad \mathit{fin}(t_c)}{x : \mathrm{ch}(\tau, !^0_{t_c}); (\Gamma_1 \mid \Gamma_2) \vdash x!!e.\, P}\ (\textsc{T-Outdlf})$$

$$\frac{\Gamma, y : \tau \vdash P}{x : \mathrm{ch}(\tau, ?^0_{t_c}); \Gamma \vdash x?y.\, P}\ (\textsc{T-In}) \qquad \frac{\Gamma, y : \tau \vdash P \quad \mathit{fin}(t_c)}{x : \mathrm{ch}(\tau, ?^0_{t_c}); \Gamma \vdash x??y.\, P}\ (\textsc{T-Indlf})$$

$$\frac{\Gamma_1 \vdash b : \mathrm{bool} \quad \Gamma_2 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1 \mid \Gamma_2 \vdash \mathrm{if}\ b\ \mathrm{then}\ P\ \mathrm{else}\ Q}\ (\textsc{T-If}) \qquad \frac{\Gamma_1 \vdash e : \tau \quad \Gamma_2, x : \tau \vdash P}{\Gamma_1 \mid \Gamma_2 \vdash \mathrm{let}\ x = e\ \mathrm{in}\ P}\ (\textsc{T-Let})$$

$$\frac{\Gamma, x : \mathrm{ch}(\tau, U) \vdash P \quad \mathit{rel}(U) \quad \tau \le \uparrow^0 \tau}{\Gamma \vdash (\nu x)\, P}\ (\textsc{T-New}) \qquad \frac{\Gamma' \vdash P \quad \Gamma \le \Gamma'}{\Gamma \vdash P}\ (\textsc{T-Weak})$$

**図 3**　型付け規則
Fig. 3　Typing rules.

**図 3** に型判定規則を示す．なお，図 3 中では，プロセス中のラベルは省略している．規則T-Outdlf とT-Indlf に課せられている条件 $\mathit{fin}(t_c)$ は，$t_c$ が有限の値（$\infty$ でない）であることを表す．主な規則は以下のとおり．

- T-New：条件 $\mathit{rel}(U)$ により，チャネル $x$ を介した送受信について，権限に見合う義務が果たされていることを課している．これにより，前節のレベルに関する条件の (2) が満たされる．条件 $\tau \le \uparrow^0 \tau$ は，送受信される値の義務レベルが 0 以上であることを要求しており，義務レベル $-\infty$ がない小林の型システム[12] では必要ない．
- T-Out：$\Gamma_1$ は送信が成功した後 $P$ においてチャネルがどのように使われるかを，$\Gamma_2$ は送信した値が受信側でどう使われるかを表す．型環境 $x : \mathrm{ch}(\tau, !^{t_o}_{t_c}); (\Gamma_1 \mid \Gamma_2)$ は，$x$ がまず送信に用いられ，その後に $\Gamma_1 \mid \Gamma_2$ によって表される送受信が起きることを表している．$(x : \mathrm{ch}(\tau, !^{t_o}_{t_c})); (\Gamma_1 \mid \Gamma_2)$ の定義から，$\Gamma_1$ と $\Gamma_2$ 中に現れる $x$ と異なるチャネルに関する送受信の義務レベルは $t_c$ よりも大きくなり，前節で述べたレベルに関する条件の (1) がこれにより満たされる．また，$ob_\alpha(U) = -\infty$ ならば $t$ の値にかかわらず $ob_\alpha(\uparrow^t U) = -\infty$ であることから，$\Gamma_1 \mid \Gamma_2$ 中で義務レベル $-\infty$ を持つ動作は，$t_c$ の値にかかわらず義務レベルが $-\infty$ となり，$x$ に対する送信にブロックされない「超越的な」動作となる．
- T-Outdlf：規則T-Outdlf とほぼ同じだが，この送信がデッドロックしないことを保証するために，権限レベル $t_c$ が有限であるという制約 $\mathit{fin}(t_c)$ を課している．

### 3.3　型システムの健全性

上で定義した型システムにおいて，$\emptyset \vdash P$ が成り立つならば，$P$ はデッドロックフリーである．小林の型システムとの唯一の違いは，義務レベルとして $-\infty$ が許されていることだが，(T-Out などの) 送受信の規則で導入される義務レベルは 0 であり，また，T-New の条件 $\tau \le \uparrow^0 \tau$ によって他のチャネルを介して送受信されるチャネルの義務レベルも 0 以上であるため，(スライスを含まない) 正規のプロセスの型判定の導出中では，$-\infty$ を義務レベルとして持つ変数は出現しない．したがって，本章の型システムの健全性は，小林の型システム[12] の健全性と同様に証明できる．

## 4.　型エラースライシング

本章では，型エラースライシングのアルゴリズムを与える．なお，簡単のために，入力とするプログラム $P$ は，(使用法表現を除いて得られる) 通常の単純型システムでは型付けされているものと仮定する．

1 章で述べたように，型エラースライシングは，以下のステップからなる．

**ステップ 1（制約生成）**　入力プロセス $P$ に対して，$\emptyset \vdash P$ が成り立つための必要十分条件を表す，使用法表現とレベルに関する制約集合を出力する．ここで $C$ 中の各制約には，その制約がプログラムのどの文面に対応するかを示すラベルが付加されている．

**ステップ 2（制約の簡約）**ステップ 1 で得られた使用法表現に関するラベル付き制約集合を，レベルに関する制約のみからなる集合 $C$ に簡約する．

**ステップ 3（充足可能性判定）**ステップ 2 で出力された制約が充足可能であるか否かを判定

する．充足可能であれば入力されたプログラムは型付け可能であり，デッドロックの可能性はないと保証されるので，その旨を出力する．充足不能であればステップ 4 に進む．

**ステップ 4（極小化）** $C$ の部分集合で，充足不能な極小の集合を求め，その集合中に含まれるラベルの集合 $L$ を求める．

**ステップ 5（プログラムスライスの生成）** ラベルの集合 $L$ 中に含まれているプログラム $P$ の部分プログラムをスライスとして出力する．

たとえば，プログラム $P_1 \mid P_2 \mid P_3$ が与えられた場合，$P_i$ から抽出された制約を $C_i$ とする．$C_1 \cup C_2$ が充足不能な極小集合であれば $P_1 \mid P_2 \mid \cdots$ を，$C_1 \cup C_3$ が充足不能な極小集合であれば $P_1 \mid \cdots \mid P_3$ を，スライスとして出力することになる．

以上の手続きのうち，ステップ 2 は小林の型推論アルゴリズムと同様に行え（ただし制約の簡約の際にラベルを正しく伝播させることに注意を要する），ステップ 3 から 5 までは既存の型エラースライシングと同様に機械的に行うことができる（ただし扱う制約が複雑なので工夫を要する）．したがって，適切なスライシングを得るために鍵となるのは，1 番目のステップにおいて各プログラムの文面にどのような制約を対応づけるかである．この点についてまず 4.1 節で詳しく議論したのち，4.2 節以降で残りのステップについて述べる．

### 4.1　ステップ 1：制約生成

制約を生成するアルゴリズムを構成するためには，まず前節の型判定規則を変形し，各プログラムの構成子に対してただ 1 つの規則が存在するようにするとともに，規則の各条件にラベルを付加する必要がある．各構成子用の型判定規則から，ラベルが付いている制約を取り除いて得られる型判定規則を「$\cdots$」の型判定規則と見なすと，本章冒頭で述べた型エラースライシングの手続きに従って出力される型エラースライスは，「型付け不能な極小のプログラムスライス」となるはずである．したがって，各型判定規則の制約をどのようにラベル付けするかによって，結果として得られる型エラースライスが左右される．

たとえば，受信に関する規則を考えよう．通常の型推論アルゴリズムの構成の際には，前章の規則T-In とT-Weak とを組み合わせて得られる以下のような規則を準備する．

$$\frac{\begin{array}{c}\Gamma \vdash P \\ \text{if } y \in \mathit{dom}(\Gamma) \text{ then } \tau \overset{l}{\leq} \Gamma(y) \text{ else } \mathit{ob}(\tau) \overset{l}{=} \infty \\ \Gamma' \overset{l}{\leq} x : \mathrm{ch}(\tau, ?^{t_o}_{t_c}); (\Gamma \setminus \{y\})\end{array}}{\Gamma' \vdash x?^{l} y.\, P}$$

しかしながら，この規則から $l$ というラベルの付いた制約を取り除き，受信部分を「$\cdots$」で置き換えると，

$$\frac{\Gamma \vdash P}{\Gamma' \vdash \cdots_{\{x\}}.P}$$

となり，プロセス $\cdots_{\{x\}}.P$ の型環境 $\Gamma'$ は何でもよいことになる．したがって，$\Gamma' = x_1 : \mathrm{ch}(\tau_1, \mu\alpha.\alpha), \ldots, x_n : \mathrm{ch}(\tau_n, \mu\alpha.\alpha)$ とすると $\Gamma'$ のすべてのチャネルについての義務レベルが $-\infty$ となり，$\cdots_{\{x\}}.P$ を含む任意のプロセスが型付けできることになってしまう（その結果として，1 章で述べた理想のスライスの例は型エラースライスではなくなり，$s$ に関する送受信が残ったものが極小のスライスとなってしまう）．

そこで，受信のための型判定規則としては以下のものを用いる．

$$\frac{\begin{array}{c}\Gamma \vdash P \\ \text{if } y \in \mathit{dom}(\Gamma) \text{ then } \tau \overset{l}{\leq} \Gamma(y) \text{ else } \tau \overset{l}{\leq} 0 \\ \mathrm{ch}(\tau, U) \overset{l}{\leq} \mathrm{ch}(\tau, ?^{t_o}_{t_c}); \Gamma(x) \\ \Gamma \setminus \{x, y\} \overset{l}{\leq} \uparrow^{t_c+1}(\Gamma \setminus \{x, y\})\end{array}}{x : \mathrm{ch}(\tau, U), \Gamma \setminus \{x, y\} \vdash x?^{l} y.\, P} \qquad \text{(TL-In)}$$

ラベルを無視すれば，この規則もT-In とT-Weak の組合せとして得られ，通常のプロセスの型判定規則としては上のものと等価である．しかしながら，ラベル付きの制約を取り除いて得られる「$\cdots$」の規則は次のようになる．

$$\frac{\Gamma \vdash P}{x : \mathrm{ch}(\tau, U), \Gamma \setminus \{x, y\} \vdash \cdots_{\{x\}}.P} \qquad \text{(TL-In-Slice)}$$

これにより，$\cdots_{\{x\}} P$ の型環境は $x$ 以外の変数については $P$ のそれと同一であるという制約は残り，プロセス $\cdots_{\{x\}} P$ が，「$x$ についてのみ万能で，それ以外のチャネルについては $P$ と同様に振る舞うプロセス」として解釈されることになる．

上と同様の型判定規則の変形をその他のプロセス構成子に対しても施すことにより，プロセスが型付けできるための必要十分条件を生成するアルゴリズムを構成することができる．制約生成アルゴリズム *MakeC* の定義を**図 4** に，*MakeC* 内で用いられる補助関数の定義を**図 5** に示す．*MakeC* は，単純型付き $\pi$ 計算の型推論が施されたプログラム $P$ を入力として受け取り，$\emptyset \vdash P$ が成立するための必要十分条件を表すラベル付き制約の集合 $C$ を返す．

$$
\begin{array}{l}
\mathit{MakeC}(P) = \\
\quad \textbf{let } (\emptyset, C) = \mathit{MakeCp}(P) \textbf{ in} \\
\quad\quad \mathit{Reduce}(C, \emptyset) \\
\\
\mathit{MakeCp}(0) = (\emptyset, \emptyset) \\
\\
\mathit{MakeCp}(x!^{c,l} v.\, P) = \\
\quad \textbf{let } (\Gamma_1, C_1) = \mathit{MakeCp}(P) \textbf{ in} \\
\quad \textbf{let } (\Gamma_2, \tau', C_2) = \mathit{MakeCv}(v) \textbf{ in} \\
\quad \textbf{let } (\Gamma_3, C_3) = \Gamma_1 \,|_l\, \Gamma_2 \textbf{ in} \\
\quad \textbf{let } \mathrm{ch}(\tau, \_) = \mathit{typeof}(x) \textbf{ in} \\
\quad \textbf{let } (\Gamma_4, C_4) = x : \mathrm{ch}(\tau, !^{t_o}_{t_c});_l (\Gamma_3) \textbf{ in} \\
\quad \textbf{if } c = 0 \textbf{ then} \\
\quad\quad (\Gamma_4, C_1 \cup \ldots \cup C_4 \cup \{\tau' \overset{l}{\leq} \tau\}) \\
\quad \textbf{else} \\
\quad\quad (\Gamma_4, C_1 \cup \ldots \cup C_4 \cup \{\tau' \overset{l}{\leq} \tau, \mathit{fin}^l(t_c)\}) \\
\\
\mathit{MakeCp}(x?^{c,l} y.\, P) = \\
\quad \textbf{let } (\Gamma_1, C_1) = \mathit{MakeCp}(P) \textbf{ in} \\
\quad \textbf{let } \mathrm{ch}(\tau, \_) = \mathit{typeof}(x) \textbf{ in} \\
\quad \textbf{let } C_2 = \\
\quad\quad \textbf{if } y \in \mathit{dom}(\Gamma_1) \textbf{ then } \{\tau \overset{l}{\leq} \Gamma_1(y)\} \\
\quad\quad \textbf{else } \{\rho \overset{l}{\leq} 0\} \text{ where } \tau = \mathrm{ch}(\_, \rho) \\
\quad \textbf{in} \\
\quad \textbf{let } (\Gamma_2, C_3) = x : \mathrm{ch}(\tau, ?^{t_o}_{t_c});_l (\Gamma_1 \setminus y) \textbf{ in} \\
\quad \textbf{if } c = 0 \textbf{ then } (\Gamma_2, C_1 \cup C_2 \cup C_3) \\
\quad \textbf{else } (\Gamma_2, C_1 \cup C_2 \cup C_3 \cup \{\mathit{fin}^l(t_c)\}) \\
\\
\mathit{MakeCp}((\nu x)^l\, P) = \\
\quad \textbf{let } (\Gamma, C) = \mathit{MakeCp}(P) \textbf{ in} \\
\quad\quad (\Gamma \setminus x, C \cup \{\mathit{rel}^l(U)\}) \\
\\
\mathit{MakeCp}(P \,|\, Q) = \\
\quad \textbf{let } (\Gamma_1, C_1) = \mathit{MakeCp}(P) \textbf{ in} \\
\quad \textbf{let } (\Gamma_2, C_2) = \mathit{MakeCp}(Q) \textbf{ in} \\
\quad \textbf{let } (\Gamma_3, C_3) = \Gamma_1 \,|_{dl}\, \Gamma_2 \textbf{ in} \\
\quad\quad (\Gamma_3, C_1 \cup C_2 \cup C_3)
\end{array}
$$

$$
\begin{array}{l}
\mathit{MakeCp}(*P) = \\
\quad \textbf{let } (\Gamma, C) = \mathit{MekeCp}(P) \textbf{ in} \\
\quad\quad (*\Gamma, C) \\
\\
\mathit{MakeCp}((\text{if } v \text{ then } P \text{ else } Q)^l) = \\
\quad \textbf{let } (\Gamma_1, C_1) = \mathit{MakeCp}(P) \textbf{ in} \\
\quad \textbf{let } (\Gamma_2, C_2) = \mathit{MakeCp}(Q) \textbf{ in} \\
\quad \textbf{let } (\Gamma_3, \_, C_3) = \mathit{MakeCv}(v) \textbf{ in} \\
\quad \textbf{let } (\Gamma_4, C_4) = \Gamma_1 \,\&_l\, \Gamma_2 \textbf{ in} \\
\quad \textbf{let } (\Gamma_5, C_5) = \Gamma_3 \,|_{dl}\, \Gamma_4 \textbf{ in} \\
\quad\quad (\Gamma_5, C_1 \cup \ldots \cup C_5) \\
\\
\mathit{MakeCp}(\text{let } x = e \text{ in } P) = \\
\quad \textbf{let } (\Gamma_1, C_1) = \mathit{MakeCp}(P) \textbf{ in} \\
\quad \textbf{let } (\Gamma_2, \tau, C_2) = \mathit{MakeCv}(e) \textbf{ in} \\
\quad \textbf{let } C_3 = \textbf{if } x \in \mathit{dom}(\Gamma_1) \\
\quad\quad \textbf{then } \{\tau \overset{dl}{\leq} \Gamma_1(x)\} \\
\quad\quad \textbf{else } \{\rho \overset{dl}{\leq} 0\} \text{ where } \tau = \mathrm{ch}(\_, \rho) \textbf{ in} \\
\quad \textbf{let } (\Gamma_3, C_4) = (\Gamma_1 \setminus x) \,|_{dl}\, \Gamma_2 \textbf{ in} \\
\quad\quad (\Gamma_3, C_1 \cup \ldots \cup C_4) \\
\\
\mathit{MakeCv}(b) = (0, \mathrm{bool}, 0) \\
\\
\mathit{MakeCv}(x^l) = \\
\quad \textbf{let } \tau = \mathit{typeof}(x) \textbf{ in} \\
\quad \textbf{let } \tau' = \mathit{typeof}(x) \textbf{ in} \\
\quad\quad (x : \tau, \tau', \{\tau \overset{l}{\leq} \tau'\}) \\
\\
\mathit{MakeCv}(\langle e_1, e_2 \rangle) = \\
\quad \textbf{let } (\Gamma_1, \tau_1, C_1) = \mathit{MakeCv}(e_1) \textbf{ in} \\
\quad \textbf{let } (\Gamma_2, \tau_2, C_2) = \mathit{MakeCv}(e_2) \textbf{ in} \\
\quad \textbf{let } (\Gamma_3, C_3) = \Gamma_1 \,|_{dl}\, \Gamma_2 \textbf{ in} \\
\quad\quad (\Gamma_3, \tau_1 \times \tau_2, C_1 \cup C_2 \cup C_3) \\
\\
\mathit{MakeCv}(\mathrm{proj}^l_i(e)) = \\
\quad \textbf{let } (\Gamma, \tau_1 \times \tau_2, C) = \mathit{MakeCv}(e) \textbf{ in} \\
\quad \textbf{let } \tau' = \mathit{typeof}(\mathrm{proj}^l_i(e)) \textbf{ in} \\
\quad\quad (\Gamma, \tau', C \cup \{\tau_i \overset{l}{\leq} \tau', \rho \overset{l}{\leq} 0\}) \\
\quad\quad (\text{where } \tau_{3-i} = \mathrm{ch}(\_, \rho))
\end{array}
$$

**図 4**　制約生成アルゴリズム
Fig. 4　A labeled type inference algorithm.

$$
\begin{array}{l}
\mathrm{bool} \,|_l\, \mathrm{bool} = (\mathrm{bool}, \emptyset) \\
\tau_{11} \times \tau_{12} \,|_l\, \tau_{21} \times \tau_{22} = \\
\quad \textbf{let } (\tau_1, C_1) = \tau_{11} \,|_l\, \tau_{21} \textbf{ in} \\
\quad \textbf{let } (\tau_2, C_2) = \tau_{12} \,|_l\, \tau_{22} \textbf{ in} \\
\quad\quad (\tau_1 \times \tau_2, C_1 \cup C_2) \\
\mathrm{ch}(\tau_1, U_1) \,|_l\, \mathrm{ch}(\tau_2, U_2) = \\
\quad (\mathrm{ch}(\tau_1, \rho), \\
\quad\quad \{\tau_1 \overset{l}{\leq} \tau_2, \tau_2 \overset{l}{\leq} \tau_1, \rho \overset{l}{\leq} U_1 \,|\, U_2\}) \\
\quad (\textit{where } \rho \textit{ is fresh}) \\
\Gamma_1 \,|_l\, \Gamma_2 = (\Gamma_1 \cup \Gamma_2, \emptyset) \\
\quad (\textit{if } \mathit{dom}(\Gamma_1) \cap \mathit{dom}(\Gamma_2) = \emptyset) \\
(\{x : \tau_1\} \cup \Gamma_1) \,|_l\, (\{x : \tau_2\} \cup \Gamma_2) = \\
\quad \textbf{let } (\tau, C_1) = \tau_1 \,|_l\, \tau_2 \textbf{ in} \\
\quad \textbf{let } (\Gamma, C_2) = \Gamma_1 \,|_l\, \Gamma_2 \textbf{ in} \\
\quad\quad (\{x : \tau\} \cup \Gamma, C_1 \cup C_2) \\
\uparrow^t_l \mathrm{bool} = (\mathrm{bool}, \emptyset) \\
\uparrow^t_l (\tau_1 \times \tau_2) = \\
\quad \textbf{let } (\tau'_1, C_1) = \uparrow^t_l \tau_1 \textbf{ in} \\
\quad \textbf{let } (\tau'_2, C_2) = \uparrow^t_l \tau_2 \textbf{ in} \\
\quad\quad ((\tau'_1 \times \tau'_2), C_1 \cup C_2) \\
\uparrow^t_l \mathrm{ch}(\tau, U) = (\mathrm{ch}(\tau, \rho), \{\rho \overset{dl}{\leq} U, \rho \overset{l}{\leq} \uparrow^t \rho\}) \\
\quad (\textit{where } \rho \textit{ is fresh}) \\
\uparrow^t_l \{x : \tau\} \cup \Gamma = \\
\quad \textbf{let } (\tau', C) = \uparrow^t_l \tau \textbf{ in} \\
\quad \textbf{let } (\Gamma', C') = \uparrow^t_l \Gamma \textbf{ in} \\
\quad\quad (\{x : \uparrow^t_l \tau\} \cup \uparrow^t_l \Gamma, C \cup C') \\
(x : \mathrm{ch}(\tau, \alpha^{t_o}_{t_c});_l \Gamma) = \\
\quad \textbf{let } (\Gamma', C) = \uparrow^{t_c+1}_l (\Gamma) \textbf{ in} \\
\quad\quad (\{x : \mathrm{ch}(\tau, \rho)\} \cup \Gamma, C \cup \{\rho \overset{l}{\leq} \alpha^{t_o}_{t_c}\}) \\
\quad (\textit{where } \rho \textit{ is fresh} \wedge x \notin \mathit{dom}(\Gamma)) \\
(x : \mathrm{ch}(\tau, \alpha^{t_o}_{t_c});_l (\{x : \mathrm{ch}(\tau', U)\} \cup \Gamma)) = \\
\quad \textbf{let } (\Gamma', C) = \uparrow^{t_c+1}_l \Gamma_3 \textbf{ in} \\
\quad\quad (\{x : \mathrm{ch}(\tau, \rho)\} \cup \Gamma, \\
\quad\quad\quad C \cup \{\tau \overset{l}{\leq} \tau', \tau' \overset{l}{\leq} \tau, \rho \overset{l}{\leq} \alpha^{t_o}_{t_c}.U\}) \\
\quad (\textit{where } \rho \textit{ is fresh})
\end{array}
$$

$$
\begin{array}{l}
\mathrm{bool} \,\&_l\, \mathrm{bool} = (\mathrm{bool}, \emptyset) \\
\mathrm{bool} \,\&_l\, 0 = (\mathrm{bool}, \emptyset) \\
\tau_{11} \times \tau_{12} \,\&_l\, \tau_{21} \times \tau_{22} = \\
\quad \textbf{let } (\tau_1, C_1) = \tau_{11} \,\&_l\, \tau_{21} \textbf{ in} \\
\quad \textbf{let } (\tau_2, C_2) = \tau_{12} \,\&_l\, \tau_{22} \textbf{ in} \\
\quad\quad (\tau_1 \times \tau_2, C_1 \cup C_2) \\
\tau_{11} \times \tau_{12} \,\&_l\, 0 = \\
\quad \textbf{let } (\tau_1, C_1) = \tau_{11} \,\&_l\, 0 \textbf{ in} \\
\quad \textbf{let } (\tau_2, C_2) = \tau_{12} \,\&_l\, 0 \textbf{ in} \\
\quad\quad (\tau_1 \times \tau_2, C_1 \cup C_2) \\
\mathrm{ch}(\tau, U_1) \,\&_l\, \mathrm{ch}(\tau, U_2) = \\
\quad (\mathrm{ch}(\tau, \rho), \{\rho \overset{l}{\leq} U_1 \,\&\, U_2\}) \\
\quad (\textit{where } \rho \textit{ is fresh}) \\
\mathrm{ch}(\tau, U_1) \,\&_l\, 0 = (\mathrm{ch}(\tau, \rho), \{\rho \overset{l}{\leq} U_1 \,\&\, 0\}) \\
\quad (\textit{where } \rho \textit{ is fresh}) \\
(\{x : \tau_1\} \cup \Gamma_1) \,\&_l\, (\{x : \tau_2\} \cup \Gamma_2) = \\
\quad \textbf{let } (\tau, C_1) = \tau_1 \,\&_l\, \tau_2 \textbf{ in} \\
\quad \textbf{let } (\Gamma, C_2) = \Gamma_1 \,\&_l\, \Gamma_2 \textbf{ in} \\
\quad\quad (\{x : \tau\} \cup \Gamma, C_1 \cup C_2) \\
(\{x : \tau\} \cup \Gamma_1) \,\&_l\, \Gamma_2 = \\
\Gamma_2 \,\&_l\, (\{x : \tau\} \cup \Gamma_1) = \\
\quad \textbf{let } (\tau', C_1) = \tau \,\&_l\, 0 \textbf{ in} \\
\quad \textbf{let } (\Gamma', C_2) = \Gamma_1 \,\&_l\, \Gamma_2 \textbf{ in} \\
\quad\quad (\{x : \tau'\} \cup \Gamma', C_1 \cup C_2) \\
\quad (\textit{where } x \notin \mathit{dom}(\Gamma_2)) \\
\mathit{Reduce}(\emptyset, C) = C \\
\mathit{Reduce}(\{\mathit{fin}^l(t)\} \uplus C', C) = \\
\quad \mathit{Reduce}(C', \{\mathit{fin}^l(t)\} \uplus C) \\
\mathit{Reduce}(\{\rho \overset{l}{\leq} U'\} \uplus C', C) = \\
\quad \mathit{Reduce}(C, \{\rho \overset{l}{\leq} U'\} \uplus C) \\
\mathit{Reduce}(\{\mathrm{bool} \overset{l}{\leq} \mathrm{bool}\} \uplus C', C) = \\
\quad \mathit{Reduce}(C', C) \\
\mathit{Reduce}(\{\langle \tau_1, \tau_2 \rangle \overset{l}{\leq} \langle \tau'_1, \tau'_2 \rangle\} \uplus C', C) = \\
\quad \mathit{Reduce}(\{\tau_1 \overset{l}{\leq} \tau'_1, \tau_2 \overset{l}{\leq} \tau'_2\} \uplus C', C) \\
\mathit{Reduce}(\{\mathrm{ch}(\tau_1, U_1) \overset{l}{\leq} \mathrm{ch}(\tau_2, U_2)\} \uplus C', C) = \\
\quad \mathit{Reduce}(\{\tau_1 \overset{l}{\leq} \tau_2, \tau_2 \overset{l}{\leq} \tau_1, U_1 \overset{l}{\leq} U_2\} \uplus C', C)
\end{array}
$$

**図 5**　補助関数の定義
Fig. 5　Auxiliary functions.

図中の定義において，*typeof* は式 $e$ を受け取り，$e$ の単純型に fresh な使用法変数を付加して返す関数である．また，$x!^l P$，$x?^l P$ をそれぞれ $x!^{0,l}.P$，$x?^{0,l}.P$ と表し，$x!!^l P$，$x??^l P$ をそれぞれ $x!^{1,l}.P$，$x?^{1,l}.P$ と表している．

*MakeC* により生成される制約は次の 3 通りのうちのいずれかである．

$$\rho \overset{l}{\leq} U \qquad rel^l(\rho) \qquad fin^l(t)$$

ここで，各制約に付加されたラベルは，その制約がプログラムのどの箇所から生じたのかを表す．$dl$ はダミーのラベルである．

### 4.2　ステップ 2：制約の簡約

本ステップでは，制約 $U_1 \overset{l}{\leq} U_2$ および $rel^l(\rho)$ をレベルの制約に還元する．まず，制約中の各使用法表現変数 $\rho$ について，レベルを表す変数 $\eta_{\rho,!,o}$，$\eta_{\rho,!,c}$，$\eta_{\rho,?,o}$，$\eta_{\rho,?,c}$ を用意する．これらは，それぞれ使用法表現 $\rho$ の送信の義務レベルと権限レベル，受信の義務レベルと権限レベルを表す．

部分使用法関係の制約 $\rho \overset{l}{\leq} U$ に対しては，以下のレベルの制約を生成する．

$$\{ob_\alpha(U) \overset{l}{\leq} \eta_{\rho,\alpha,o},\quad \eta_{\rho,\alpha,c} \overset{l}{\leq} cap_\alpha(U) \mid \alpha \in \{!,?\}\}$$

ただし，$ob_\alpha(U)$ および $cap_\alpha(U)$ は，図 2 の義務レベルと権限レベルの定義において，変数に対する定義を $cap_\alpha(\rho) = \eta_{\rho,\alpha,c}$ および $ob_\alpha(\rho) = \eta_{\rho,\alpha,o}$ で置き換えたものとする．

次に，制約 $rel^l(\rho)$ のレベル制約への還元を定義するために，使用法表現のラベル付遷移関係 $U \overset{L}{\rightarrow}{}^*_C\, U'$ を以下の規則によって定義する．

$$\frac{U \to U'}{U \overset{dl}{\rightarrow}_C U'} \qquad \frac{(U \overset{l}{\leq} U') \in C}{U \overset{l}{\rightarrow}_C U'}$$

$$\frac{U_1 \overset{l}{\rightarrow}_C U_1'}{U_1 \,|\, U_2 \overset{l}{\rightarrow}_C U_1' \,|\, U_2}$$

$$U \overset{\emptyset}{\rightarrow}{}^*_C U \qquad \frac{U \overset{l}{\rightarrow}_C U'' \quad U'' \overset{L}{\rightarrow}{}^*_C U'}{U \overset{\{l\}\cup L}{\rightarrow}{}^*_C U'}$$

上の関係を用い，$rel^l(\rho)$ を以下の制約で置き換える．

$$\left\{ ob_?(U) \overset{L}{\leq} cap_!(U), ob_!(U) \overset{L}{\leq} cap_?(U) \mid \rho \overset{L}{\rightarrow}{}^*_C U \right\}$$

**注 4.1**　$\rho \overset{L}{\rightarrow}{}^*_C U$ を満たす $U$ は実際には無限集合である可能性がある．しかしながら，義務レベルや権限レベルが使用法の重複に依存しない（$ob_\alpha(U \,|\, U) = ob_\alpha(U)$ が成り立つ）ことから，$\rho$ から到達可能な遷移先としては，$U \,|\, U \sim U$（および $|$ と 0 に関するモノイド規則）によって定まる同値類を考えればよく，$\{ob_?(U) \overset{L}{\leq} cap_!(U), ob_!(U) \overset{L}{\leq} cap_?(U) \mid \rho \overset{L}{\rightarrow}{}^*_C U\}$ は有限集合となる．

本ステップの操作は，小林らによるデッドロック解析[11] で提案された，制約 $rel(U)$ からレベルに関する制約への簡約をペトリネットの到達可能性問題に帰着する手法に，遷移に用いたラベルの集合 $L$ を求める拡張を施すことで行うことができる．

以上の変換によって，ステップ 1 で得られた制約集合は，レベルに関する不等式 $\eta_i \overset{L}{\geq} f(\eta_1, \ldots, \eta_n)$ と $fin^l(\eta_i)$ の形の制約に簡約される．ただし，$f(\eta_1, \ldots, \eta_n)$ はレベル変数 $\eta_1, \ldots, \eta_n$ および $\mathbf{max}, \mathbf{min}, \uparrow(\cdot,\cdot)$ のみから構成されるレベル式である（$\uparrow(t_1, t_2)$ は，$t_2 = -\infty$ ならば $-\infty$，$t_2 \neq -\infty$ ならば $\mathbf{max}(t_1, t_2)$ を返す単調関数を表す）．

### 4.3　ステップ 3，4：充足可能性判定と極小化

制約集合の充足可能性の判定アルゴリズムを与えれば，Haack らの手法[8] や Ueda らの手法[4),16),17] などの既存の手法を用いて充足不能な極小集合を求めることができる（現在の実装においては，Haack らの手法を用いている）．そこで，以下では，ステップ 2 で得られたレベルに関する制約集合の充足可能性判定アルゴリズムのみを議論する．

レベルに関する制約集合の充足可能性判定は，小林による情報流解析のための型推論[11] におけるそれと同様の手法で行う．異なる点は，小林による情報流解析のための型推論[11] ではレベル変数のとりうる値の範囲を $\mathbf{Nat} \cup \{\infty\}$ であるのに対し，本論文では，レベル変数の値として $-\infty$ が加わっていることである．そこで，まずレベル変数のとりうる値を $-\infty$ と，それ以外を抽象化した値 $\infty$ とに抽象化したうえで，（抽象化された）レベルの制約を解き，値 $-\infty$ をとるレベル変数を求める．これは，制約の形が $\vec{\eta} \geq \vec{f}(\vec{\eta})$ であり，$\vec{f}$ は単調関数，$\vec{\eta}$ のとりうる値の集合が有限であることから，標準的な制約解消アルゴリズムによって解くことができる[14]．値 $-\infty$ をとるレベル変数が定まった後は，残りの変数のとりうる値は $\mathbf{Nat} \cup \{\infty\}$ であるため，小林のアルゴリズム[11] を適用すればよい．

### 4.4　スライシング

前節のアルゴリズムにより求まったラベル集合 $L$ から，プログラムスライスを求めるアルゴリズム *Slice* を**図 6** に示す．このアルゴリズムはプロセスとラベルの集合を入力とし，スライスを出力とする．スライスはデッドロックの原因箇所として求めたラベルの集合に含まれている部分を残し，それ以外の部分を *dots* で置き換えたものである．

$$
\begin{array}{l}
\mathit{ESlice}(b, L) = \mathit{dots} \\
\\
\mathit{ESlice}(x^l, L) = \\
\quad \textbf{if } l \in L \textbf{ then } x \textbf{ else } \mathit{dots} \\
\\
\mathit{ESlice}(\langle e_1, e_2 \rangle, L) = \\
\quad \textbf{let } e_1' = \mathit{ESlice}(e_1, L) \textbf{ in} \\
\quad \textbf{let } e_2' = \mathit{ESlice}(e_2, L) \textbf{ in} \\
\quad \textbf{if } e_1' = \mathit{dots} \wedge e_2' = \mathit{dots} \textbf{ then } \mathit{dots} \\
\quad \textbf{else } \langle e_1', e_2' \rangle \\
\\
\mathit{ESlice}(\mathrm{proj}_i^l(e), L) = \\
\quad \textbf{let } e' = \mathit{ESlice}(e, L) \textbf{ in} \\
\quad \textbf{if } e' = \mathit{dots} \vee l \notin L \textbf{ then } \mathit{dots} \\
\quad \textbf{else } \mathrm{proj}_i(e') \\
\\
\mathit{Slice}(\mathbf{0}^l, L) = \\
\quad \textbf{if } l \in L \textbf{ then } 0 \\
\quad \textbf{else } \mathit{dots} \\
\\
\mathit{Slice}(x!^l e.\, P, L) = \\
\quad \textbf{let } e' = \mathit{ESlice}(e, L) \textbf{ in} \\
\quad \textbf{let } P' = \mathit{Slice}(P, L) \textbf{ in} \\
\quad \textbf{if } l \in L \textbf{ then } x!e'.\, P' \\
\quad \textbf{else } P' \\
\\
\mathit{Slice}(x?^l y.\, P, L) = \\
\quad \textbf{let } P' = \mathit{Slice}(P, L) \textbf{ in} \\
\quad \textbf{if } l \in L \textbf{ then } x?y.\, P' \\
\quad \textbf{else } P' \\
\\
\mathit{Slice}((\nu x)^l\, P, L) = \\
\quad \textbf{let } P' = \mathit{Slice}(P, L) \textbf{ in} \\
\quad \textbf{if } l \in L \textbf{ then } (\nu x)\, P' \\
\quad \textbf{else } P'
\end{array}
\qquad
\begin{array}{l}
\mathit{Slice}(P \mid Q, L) = \\
\quad \textbf{let } P' = \mathit{Slice}(P, L) \textbf{ in} \\
\quad \textbf{let } Q' = \mathit{Slice}(Q, L) \textbf{ in} \\
\quad \textbf{if } (P' = \mathit{dots}) \wedge (Q' = \mathit{dots}) \\
\quad \textbf{then } \mathit{dots} \\
\quad \textbf{else if } (P' = \mathit{dots}) \textbf{ then } Q' \\
\quad \textbf{else if } (Q' = \mathit{dots}) \textbf{ then } P' \\
\quad \textbf{else } P' \mid Q' \\
\\
\mathit{Slice}(*P, L) = \\
\quad \textbf{let } P' = \mathit{Slice}(P, L) \textbf{ in} \\
\quad \textbf{if } (P' = \mathit{dots}) \textbf{ then } \mathit{dots} \\
\quad \textbf{else } *P' \\
\\
\mathit{Slice}((\textbf{if } v \textbf{ then } P \textbf{ else } Q)^l, L) = \\
\quad \textbf{let } v' = \mathit{ESlice}(v, L) \textbf{ in} \\
\quad \textbf{let } P' = \mathit{Slice}(P, L) \textbf{ in} \\
\quad \textbf{let } Q' = \mathit{Slice}(Q, L) \textbf{ in} \\
\quad \textbf{if } l \in L \\
\quad \textbf{then } \textbf{if } v' \textbf{ then } P' \textbf{ else } Q' \\
\quad \textbf{else} \\
\quad \textbf{if } (P' = \mathit{dots}) \wedge (Q' = \mathit{dots}) \\
\quad \textbf{then } \mathit{dots} \\
\quad \textbf{else if } (P' = \mathit{dots}) \textbf{ then } Q' \\
\quad \textbf{else if } (Q' = \mathit{dots}) \textbf{ then } P' \\
\quad \textbf{else } \mathit{dots} \\
\\
\mathit{Slice}(\textbf{let } x = e \textbf{ in } P, L) = \\
\quad \textbf{let } e' = \mathit{ESlice}(e, L) \textbf{ in} \\
\quad \textbf{let } P' = \mathit{Slice}(P, L) \textbf{ in} \\
\quad \textbf{if } (P' = \mathit{dots}) \\
\quad \textbf{then } \mathit{dots} \\
\quad \textbf{else } \textbf{let } x = e' \textbf{ in } P'
\end{array}
$$

**図 6**　スライシングのアルゴリズム
Fig. 6　Slicing.

### 4.5　スライシングの例

**例 4.1**　以下のプロセスに対するスライシングを通して義務レベル $-\infty$ の役割を説明する．

$$(\nu x)^{l_5}\,(\nu y)^{l_6}\,(x?^{l_1} z.\, y!^{l_2} z \mid y??^{l_3} z.\, x!^{l_4} z)$$

これは $x$ と $y$ の送受信の依存関係が循環してデッドロックしているため，期待するスライスは引数の部分のみを削除して得られる以下のプロセスである．

$$(\nu x)^{l_5}\,(\nu y)^{l_6}\,(x?^{l_1} z.\, y!^{l_2} \cdots \mid y??^{l_3} z.\, x!^{l_4} \cdots)$$

簡単のため，以下ではレベル以外の使用法表現についてはあらかじめ定まっているものとすると，上のプロセスに対する型導出は**図 7** のようになる．レベルに関する制約は以下のとおり．

$$
\begin{array}{llll}
\eta_4 \overset{l_5}{\geq} \eta_1 & \eta_2 \overset{l_5}{\geq} \eta_3 & \eta_8 \overset{l_6}{\geq} \eta_5 & \eta_6 \overset{l_6}{\geq} \eta_7 \\
\eta_3 \overset{l_1}{\geq} 0 & \eta_5 \overset{l_1}{\geq} \uparrow(\eta_4 + 1, \eta_5') & \eta_5' \overset{l_2}{\geq} 0 & \\
\eta_7 \overset{l_3}{\geq} 0 & \eta_1 \overset{l_3}{\geq} \uparrow(\eta_8 + 1, \eta_1') & \eta_1' \overset{l_4}{\geq} 0 & \mathit{fin}^{l_3}(\eta_8)
\end{array}
$$

最初の行の制約が $(\nu x)\,(\nu y)$ の部分にともなう制約 $\mathit{rel}(!^{\eta_1}_{\eta_2} \mid ?^{\eta_3}_{\eta_4})$ と $\mathit{rel}(!^{\eta_5}_{\eta_6} \mid ?^{\eta_7}_{\eta_8})$ から得られるレベル制約である．次の行が $x?^{l_1} z.\, y!^{l_2} z$ の部分から得られる制約，最後の行が $y??^{l_3} z.\, x!^{l_4} z$ の部分から得られる制約である．制約全体が充足不能であることは，

$$
\begin{array}{l}
\eta_8 \overset{l_6}{\geq} \eta_5 \overset{l_1}{\geq} \uparrow(\eta_4 + 1, \eta_5') \overset{l_2}{\geq} \eta_4 + 1 \\
\overset{l_5}{\geq} \eta_1 + 1 \overset{l_3}{\geq} \uparrow(\eta_8 + 1, \eta_1') + 1 \overset{l_4}{\geq} (\eta_8 + 1) + 1
\end{array}
$$

から $\eta_8 = \infty$ であり，$\mathit{fin}^{l_3}(\eta_8)$ が満たされないことから分かる（$\uparrow(\eta_4 + 1, \eta_5') \overset{l_2}{\geq} \eta_4 + 1$ は，$\eta_5' \overset{l_2}{\geq} 0$ から導かれる）．

一方，上の制約から，ラベル $l_1, \ldots, l_6$ のどれを取り除いた制約も充足可能である．たとえば，$l_2$ の制約 $\eta_5' \overset{l_2}{\geq} 0$ を除いた場合，$\eta_5' = \eta_5 = \eta_8 = -\infty$ とし，その他のレベルをすべて 0 とすれば上記の制約が満たされる．したがって，充足不能な極小集合に対応するラベルは $\{l_1, \ldots, l_6\}$ となり，期待するスライス

$$(\nu x)^{l_5}\,(\nu y)^{l_6}\,(x?^{l_1} z.\, y!^{l_2} \cdots \mid y??^{l_3} z.\, x!^{l_4} \cdots)$$

が得られる．

なお，小林の型システム[12] のように義務レベルのとりうる値がつねに 0 以上である場合には，上の制約のうち，$\eta_5' \overset{l_2}{\geq} 0$ と $\eta_1' \overset{l_4}{\geq} 0$ がなくても充足不能となり，得られるスライスは

$$(\nu x)^{l_5}\,(\nu y)^{l_6}\,(x?^{l_1} z.\, \cdots \mid y??^{l_3} z.\, \cdots)$$

$$\dfrac{\dfrac{x:\mathrm{ch}(\mathrm{bool},0),y:\mathrm{ch}(\mathrm{bool},!^{\eta_5'}_{\eta_6'})\vdash y!^{l_2}z}{x:\mathrm{ch}(\mathrm{bool},?^{\eta_3}_{\eta_4}),y:\mathrm{ch}(\mathrm{bool},!^{\eta_5}_{\eta_6})\vdash x?^{l_1}z.\,y!^{l_2}z}\quad\dfrac{x:\mathrm{ch}(\mathrm{bool},!^{\eta_1'}_{\eta_2'}),y:\mathrm{ch}(\mathrm{bool},0)\vdash x!^{l_4}z}{x:\mathrm{ch}(\mathrm{bool},!^{\eta_1}_{\eta_2}),y:\mathrm{ch}(\mathrm{bool},?^{\eta_7}_{\eta_8})\vdash y??^{l_3}z.\,x!^{l_4}z}}{\dfrac{x:\mathrm{ch}(\mathrm{bool},!^{\eta_1}_{\eta_2}\mid?^{\eta_3}_{\eta_4}),y:\mathrm{ch}(\mathrm{bool},!^{\eta_5}_{\eta_6}\mid?^{\eta_7}_{\eta_8})\vdash x?^{l_1}z.\,y!^{l_2}z\,|\,y??^{l_3}z.\,x!^{l_4}z}{x:\mathrm{ch}(\mathrm{bool},!^{\eta_1}_{\eta_2}\mid?^{\eta_3}_{\eta_4})\vdash(\nu x)^{l_5}\,(\nu y)^{l_6}\,(x?^{l_1}z.\,y!^{l_2}z\,|\,y??^{l_3}z.\,x!^{l_4}z)}}{\emptyset\vdash(\nu x)^{l_5}\,(\nu y)^{l_6}\,(x?^{l_1}z.\,y!^{l_2}z\,|\,y??^{l_3}z.\,x!^{l_4}z)}$$

**図 7**　$(\nu x)^{l_5}\,(\nu y)^{l_6}\,(x?^{l_1}z.\,y!^{l_2}z\,|\,y??^{l_3}z.\,x!^{l_4}z)$ に対する導出
Fig. 7　Derivation for $(\nu x)^{l_5}\,(\nu y)^{l_6}\,(x?^{l_1}z.\,y!^{l_2}z\,|\,y??^{l_3}z.\,x!^{l_4}z)$.

となってしまう．

**例 4.2**　1 章で用いた以下のプロセス $P$ について，スライシングの例を示す．

$(\nu s)^{l_1}\,(\nu r_1)^{l_2}\,(\nu r_2)^{l_3}$
$s?^{l_4}b.\,(\text{if } b \text{ then } r_1?^{l_5}x.\,r_2!^{l_6}x \text{ else } r_2!^{l_7}0)^{l_8}$
$\mid s!^{l_9}\mathrm{true}.\,r_2??^{l_{10}}y.\,r_1!^{l_{11}}y$

このプロセス $P$ を $MakeC$ に与えて得られる制約の集合は以下の $C$ となる（簡単のために，一部制約を省略し，複数の部分使用法制約を 1 つにまとめて表した）．

$$C = C_1 \cup C_2 \cup C_3 \cup \cdots$$
$$C_1 = \left\{ \rho_s \overset{\{l_4,l_9\}}{\leq} ?^{t_0}_{t_1}|!^{t_8}_{t_9} \right\}$$
$$C_2 = \left\{ \begin{array}{l} \rho_{r_1} \overset{\{l_5,l_8\}}{\leq} (?^{t_2}_{t_3}\&0)|\rho_{r_1r}, \\ \rho_{r_1r} \overset{l_{10}}{\leq} \uparrow^{t_{11}+1} \rho_{r_1r} \\ \rho_{r_1r} \overset{l_{11}}{\leq} !^{t_{12}}_{t_{13}} \end{array} \right\}$$
$$C_3 = \left\{ \begin{array}{l} \rho_{r_2} \overset{\{l_8,l_{10}\}}{\leq} (\rho_{r_{2lt}}\&\rho_{r_{2le}})|?^{t_{10}}_{t_{11}}, \\ \rho_{r_{2lt}} \overset{l_5}{\leq} \uparrow^{t_3+1} \rho_{r_{2lt}}, \\ \rho_{r_{2lt}} \overset{l_6}{\leq} !^{t_4}_{t_5}, \\ \rho_{r_{2le}} \overset{l_7}{\leq} !^{t_6}_{t_7}, \\ \mathit{fin}^{l_{10}}(t_{11}) \end{array} \right\}$$

ここで，$C_1$ は $P$ 中のチャネル $s$ の使用法に関する制約，$C_2$ は $r_1$ の，$C_3$ は $r_2$ の使用法に関する制約である．$C_2$ 中の制約 $\rho_{r_1r} \overset{l_{10}}{\leq} \uparrow^{t_{11}+1} \rho_{r_1r}$ は $l_{10}$ でラベル付けされた $r_2$ に対する通信を行うプロセスについて制約を生成するときに生成された制約で，$l_{11}$ でラベル付けされた通信の義務レベルが，$l_{10}$ でラベル付けされた通信の権限レベルよりも大きいことを表している．$C_3$ 中の制約 $\rho_{r_{2lt}} \overset{l_5}{\leq} \uparrow^{t_3+1} \rho_{r_{2lt}}$ も同様に通信間の依存関係によって生ずるレベル間の大小関係を表す．

上記の制約をステップ 2 の手続きで簡約し，極小化すると，ラベルの集合

$\{l_2, l_3, l_5, l_6, l_8, l_{10}, l_{11}\}$

が得られる．このラベル集合を基にスライシングを行うと，以下のスライス

$\cdots(\text{if } \cdots \text{ then } r_1?x.\,r_2!x \text{ else } \cdots)$
$\mid \cdots r_2??y.\,r_1!y$

を得る．

ここで，4.1 節で述べたような型付け規則の変換を行うことなく，小林らの型システム[12]について単純に型エラースライシングを適用するようなラベル付けを行った場合，$l_4$ と $l_9$ でラベル付けされた通信が，その後に通信に使われるチャネル $r_1$ と $r_2$ の型に影響を及ぼす．そのため，$C_2$ 中の $\rho_{r_1} \overset{\{l_5,l_8\}}{\leq} (?^{t_2}_{t_3}\&0)|\rho_{r_1r}$ にラベル $l_4$，$l_9$，$l_{10}$ が付加され，$C_3$ 中の $\rho_{r_2} \overset{\{l_8,l_{10}\}}{\leq} (\rho_{r_{2lt}}\&\rho_{r_{2le}})|?^{t_{10}}_{t_{11}}$ にラベル $l_4$，$l_9$ が付加される．その結果，極小化後のラベル集合にも $l_4$，$l_9$ が付加され，本来は不要な $s$ に対する通信がスライス中に残ってしまう．すなわち，デッドロックすると判定された通信よりも前に行われる通信は，すべてスライス中に残ることになる．

## 5. 実　　験

本章では，小林の `TyPiCal` を本研究の型エラースライシングで拡張し，実験を行った結

果について述べる．

**表 1** に，実験に用いたサンプルプログラムと，それらに対する出力を示す．例として，TyPiCal の配布版に同梱されていたサンプルプログラムを用いた．以下，各入力プログラムと得られたスライスについて述べる．

- simple2.pi：チャネル $x$ とチャネル $y$ について，通信の順序が 2 つのプロセスの間で一致していないためにデッドロックするプログラムである．出力では，送受信されている値が除去されており，$x$ と $y$ の相互依存によりデッドロックすることが示されている．
- simple7.pi：$x$ に対して受信を行うプロセスと，$x$ に対する受信を分岐の片方のみで行うプロセスが並行に実行されているプログラムである（分岐の条件がつねに真なので，このプログラムは実行時にデッドロックすることはないが，型システムは分岐のどちらも実行されうると見なすので，デッドロックの可能性が報告される）．出力においては，2 つ目のプロセスの else 節が 0 となっていることで，$x$ に対する送信が行われない可能性があるということが示されている．
- lock.pi：このプログラムにおいては，チャネル *secret* とチャネル *public* で変数を，チャネル $x$ とチャネル $y$ でロックをエンコードしている．1 つ目のプロセスは，変数 *secret* に代入されている値が真ならば，ロック $x$ を獲得し解放したうえで，変数 *public* に **true** を代入する．2 つめのプロセスは，ロック $x$ と $y$ をこの順番で獲得し，2 つとも解放する．ただし，ロック $x$ は初めは解放された状態であり，ロック $y$ は初めは獲得された状態である．

  このプログラム中においては，#1 と#2 で示した 2 つの通信が，デッドロックを許さない通信としてマークされている．我々の試作器では，この 2 つの通信それぞれについて，デッドロックの原因を示すスライスが出力される．#2 についてのスライスからは，$y$ が初め獲得された状態であるために，$y$ の獲得操作が成功しないことが分かる．#1 についてのスライスからは，2 つ目のプロセスが 1 つ目のプロセスよりも先にロックを獲得してしまうと，2 つ目のプロセスの $y$ の獲得操作が成功しないために，$x$ が解放されず，1 つ目のプロセスの $x$ の獲得が成功しないことが読み取れる．
- client-server-wrong.pi：1 つ目のプロセスは，チャネルを受け取り，そのチャネルに対して分岐の条件が **true** の場合にのみ返信を行うプロセスである（分岐の条件はつねに真なので，実行時には必ず返信が行われるが，simple7.pi の説明で述べたのと同様，型システムは返信が行われない可能性を考慮する）．2 つ目と 3 つ目のプロセスは，サーバにチャネルを送り，そのチャネルに対する受信を試みるプロセスで，2 つ目のプロセスにおいては，その受信についてデッドロックを許されないとマークされている．この入力に対するスライスからは，サーバ内で返信を行わない可能性があることが，else 節が 0 になっていることから読み取れる．

**表 1**　入力プログラムと，それに対して得られた型エラースライス
Table 1　Input programs and the obtained slices.

| 入力 | | 出力 |
|---|---|---|
| simple2.pi | new x in new y in<br>x??z.y!z \| y?z.x!z | new x in new y in<br>x??...y!... \| y?...x!... |
| simple7.pi | new x in<br>x??y \| if true then x!1 else 0 | new x in<br>x??y \| if .. then .. else 0 |
| lock.pi | new secret in new lock_x in<br>(new cont in<br>secret?b.(secret!b \|<br>if b then lock_x??$^{\#1}$z.<br>(lock_x!() \| cont!())<br>else 0) \|<br>cont?z.public!true) \|<br>lock_x?z.lock_y??$^{\#2}$z.<br>(lock_x!() \| lock_y!()) \|<br>lock_x!() \|<br>secret!true | #1<br>new lock_y in....new lock_x in<br>....(if .. then lock_x??.....else..) \|<br>lock_x?...lock_y?...lock_x!..... \|<br>lock_x!.....<br>#2<br>new lock_y in<br>..... \| ..lock_y??..... |
| client<br>-server<br>-wrong.pi | new server in<br>∗server?r.<br>if true then r!1 else 0<br>\| new r in server!r.r??z.print!z<br>\| new r in server!r.r?z.print!z | ....∗server?...<br>if .. then .. else 0<br>\| new r in server!r.r??..... \| .... |

現状の解析器においては，simple2.pi や lock.pi のように，通信の行われる順序がデッドロックに関わる場合には，その通信に関わる部分がすべて表示され，どのような順序で通信が起こればデッドロックするかといった情報までは表示されない．この点に関する改良は，今後の課題である．

また，現状の試作器では，スライシングのオーバヘッドが大きい．これは，試作器の実装がナイーブであることもあるが，最も大きな原因は $rel^l(\rho)$ の簡約において，TyPiCal と異なり，$\rho \xrightarrow[C]{L}{}^{*} U$ を満たす $U$ （の同値類）の集合を求める際にラベルの集合 $L$ も同時に求めなければならず，状態探索空間が爆発するためと思われる．この部分の効率化もまた今後の課題の 1 つである．

## 6. 関連研究

すでに述べたように，関数型言語を対象とした型エラースライシングは Dinesh ら[5] や Haack ら[8] により提案されている．1 章で述べたように，これらの手法を本論文で扱ったデッドロックのための型エラースライシングに単純に適用することはできない．

Ueda ら[4),16),17)] は並行論理型言語のモード解析を対象に，誤り個所を特定し自動修正を行う方法を提案している．プログラムからモード制約を取出し，充足不能な極小集合を求めてそれに対応する個所を誤り個所と判断するという点で，基本的考え方は型エラースライシングと同様である．対象言語は並行言語であるが，彼らのモードの誤り個所特定においては，本論文で扱ったデッドロック解析とは違い，異なる通信チャネル間の通信の依存関係が関わってこない．そのため，型（モード）エラースライシングのアルゴリズムを構成するうえで，本論文で行ったような（義務レベルとして $-\infty$ を導入したり，型判定規則を構成し直したりするような）基となる型システム（またはモードシステム）自体の大幅な変更は必要ないものと思われる．

並行プログラムに対する型エラースライシングとしては，レース解析のための手法がすでに提案されている[18]．その研究で使用している型システムは本研究のものと類似している（使用法表現から義務レベルと権限レベルを取り除いて得られる）が，レース解析の場合にはスライス後の *dots* をプロセス 0 と同一視してもよいという点が本論文の型エラースライシングの手法と大きく異なる．結果として，本論文と異なり，基となる型システムを大幅に変更することなしに型エラースライシングのための型システムを構成することができる．

## 7. 結　　論

$\pi$ 計算のプロセスのデッドロックの原因箇所を静的に特定するための型エラースライシングの手法を提案した．また，小林の TyPiCal に実装を組み込み，いくつかの小さなサンプルプログラムについて実験を行い，適切なスライスが得られることを確認した．

今後の課題として，スライシングアルゴリズムの正当性の厳密な議論があげられる．$dots_S$ の型判定規則を適切に与えた場合，我々のアルゴリズムは，元のプログラムから得られるスライスのうち，型付け不可能な極小のスライスを出力すると予想されるが，$dots_S$ の厳密な型判定規則の定義を含め，正当性の定式化および証明が必要である．また，5 章で述べたように，実用化のためには，スライシングアルゴリズムの効率化が必要である．

**謝辞**　本研究は科研費（20240001，17300003）の助成を受けたものです．貴重なコメントをいただいた査読者の皆様と東北大学小林・住井研究室の皆様に感謝いたします．

## 参 考 文 献

1) Abadi, M., Flanagan, C. and Freund, S.N.: Type-Based Race Detection for Java, *Programming Language Design and Implementation* (*PLDI*), pp.219–232 (2000).
2) Boyapati, C., Lee, R. and Rinard, M.: Ownership Types for Safe Proguramming: Preventing Data Races and Deadlocks, *Object-Oriented Programming, Systems, Languages and Applications* (*OOPSLA*), pp.211–230 (2002).
3) Boyapati, C. and Rinard, M.: A Parametarized Type System for Race-Free Java Programs, *Object-Oriented Programming, Systems, Languages and Applications* (*OOPSLA*), pp.56–69 (2001).
4) Cho, K. and Ueda, K.: Diagnosing Non-Well-Moded Concurrent Logic Programs, *Joint International Conference and Symposium on Logic Programming*, pp.215–229 (1996).
5) Dinesh, T.B. and Tip, F.: A slicing-based approach for locating type errors, *Proc. USENIX conference on Domain-Specific Languages*, pp.77–88 (1997).
6) Flanagan, C. and Abadi, M.: Object Types against Races, *CONCUR'99*, LNCS, Vol.1664, pp.288–303, Springer (1999).
7) Flanagan, C. and Abadi, M.: Types for Safe Locking, *Proc. ESOP 1999*, Lecture Notes in Computer Science, Vol.1576, pp.91–108 (1999).
8) Haack, C. and Wells, J.B.: Type Error Slicing in Implicitly Typed Higher-Order Languages, *SCIPROG: Science of Computer Programming*, Vol.50, No.1-3, pp.189–224 (2004).
9) Kobayashi, N.: TyPiCal: A Type-Based Static Analyzer for the Pi-Calculus. Tool available at http://www.kb.ecei.tohoku.ac.jp/koba/typical/
10) Kobayashi, N.: Type Systems for Concurrent Programs, *Proc. UNU/IIST 20th Anniversary Colloquium*, Lecture Notes in Computer Science, Vol.2757, pp.439–453, Springer (2003).
11) Kobayashi, N.: Type-Based Infomation Flow Analysis for the Pi-Caluculus, *Acta Infomatica*, Vol.42, No.4-5, pp.291–347 (2005).
12) Kobayashi, N.: A New Type System for Deadlock-Free Processes, *Proc. CONCUR 2006*, LNCS, Vol.4137, pp.233–247, Springer (2006).
13) Kobayashi, N.: Type Systems for Concurrent Programs http://www.kb.ecei.tohoku.ac.jp/~koba/papers/tutorial-type-extended.pdf (2007). Preliminary version appeared in Proceedings of UNU/IIST 20th Anniversary Colloquium, LNCS 2757, pp.439–453, Springer.
14) Rehof, J. and Mogensen, T.: Tractable Constraints in Finite Semilattices, *Science of Computer Programming*, Vol.35, No.2, pp.191–221 (1999).

15) Sumii, E. and Kobayashi, N.: A Generalized Deadlock-Free Process Calculus, *Proc. Workshop on High-Level Concurrent Language* (*HLCL'98*), ENTCS, Vol.16(3), pp.55–77 (1998).
16) Yasuhiro, A. and Ueda, K.: Kima: an Automated Error Correction System for Concurrent Logic Programs, *Automated Software Engineering*, Vol.9, No.1, pp.67–94 (2002).
17) Yasuhiro, A., Ueda, K. and Cho, K.: Error-correcting Source Code, *Principles and Practice of Constraint Programming*, pp.40–54 (1998).
18) 飯村枝里，小林直樹，末永幸平：型に基づくレース解析とその結果表示のためのスライシング，第 9 回プログラミングおよびプログラミング言語ワークショップ（PPL2007），pp.156–172 (2007).

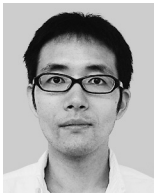
(平成 20 年 2 月 19 日受付)
(平成 20 年 5 月 1 日採録)

**飯村　枝里**

1983 年生まれ．2006 年東北大学工学部卒業．2008 年東北大学大学院情報科学研究科修士課程修了．同年オリンパス株式会社入社．

**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．2001 年 IFIP TC2 Manfred Paul Award，2003 年日本 IBM 科学賞受賞．



**末永　幸平**

1979 年生．2005 年東京大学大学院情報理工学系研究科コンピュータ科学専攻修士課程修了．2008 年同博士課程修了．博士（情報理工学）．2008 年より日本学術振興会特別研究員（PD）．並行プログラムの検証に関する研究に従事．ACM，日本ソフトウェア科学会各会員．