# 型システムを用いたロックフリースタックの検証

佐藤 駿太朗　住井 英二郎

ロックフリー性とは，デッドロックやライブロックが起きないという，並行システムにおいて重要な性質である．ロックフリー性を検証する方法として，自然言語による非形式的な証明や，紙の上ないし定理証明支援器上の形式的な証明の他に，軽量な形式手法とされる型システムに基づく手法があり，その実装として TyPiCal [Kobayashi] がある．

本研究では，TyPiCal を再帰型を持つセル構造で拡張して，Treiber [1986] のロックフリースタックを表現する．TyPiCal だけでは再帰的なプロセスのライブロックフリー性を検証できないため，TyPiCal でデッドロックフリー性を検証し，Kobayashi と Sangiorgi [2008] のハイブリッド型システムと組み合わせてロックフリー性を証明する．その際，TyPiCal では再帰型を持つデータ構造に対する操作はプリミティブ関数として拡張する必要がある．それにともない新たに発生する制約の生成と解消を実装する．加えて，無限回のスタック操作を並行に行う場合，ロックフリー性の定義の違いから生じる問題について考察する．

## 1　序論

### 1.1　背景

並行プログラミングでは，複数のスレッドから共有資源にアクセスする際，競合状態を避けるために同期制御を行う必要がある．同期にはブロッキング同期とノンブロッキング同期がある [5, chapter 1]．ブロッキング同期とは，一つのスレッドの処理を進めるために，他のスレッドを遅延させる同期である [5, chapter 1]．一般に排他制御を用いた同期はこれに含まれる．明示的にロックを用いたブロッキング同期は，ロックの取得・解放のタイミングや，複数のスレッドのスケジューリングによってはデッドロックやライブロックを起こす．ここでデッドロックとは，完了してほしい処理が完了せずに，すべてのスレッドの進行が停止することをいう．またライブロックとは，少なくとも一つのスレッドが進行し続けているにも関わらず，完了してほしい処理が完了しないことをいう．

一方，排他制御を用いない，ノンブロッキング同期も研究されている．ノンブロッキング同期は，あるスレッドの遅延や失敗が，他のスレッドの実行を妨害しないという性質（ノンブロッキング性）を満たす同期である [5, chapter 3]．ノンブロッキング性を満たしつつ共有データ構造を操作するアルゴリズムをノンブロッキングアルゴリズムと言い，各操作について保証される性質の弱い順にオブストラクションフリー，ロックフリー，ウェイトフリーの 3 種類に分類できる [5, chapter 3]．

- オブストラクションフリー
  ある操作がオブストラクションフリーであるとは，競合するスレッドをすべて停止させた状態で，任意の一つのスレッドのみを実行した場合に，有限ステップで完了するということである．
- ロックフリー
  ある操作がロックフリーであるとは， 複数のスレッドがいかなるスケジューリングをされても，それらで実行されているメソッドのうち少なくとも一つが， 有限ステップで完了するということである．
- ウェイトフリー
  ある操作がウェイトフリーであるとは，複数のス

Shuntaro Sato, Eijiro Sumii, 東北大学大学院情報科学研究科, Graduate School of Infomation Sciences, Tohoku University.

レッドがいかなるスケジューリングをされても，それらで実行されているメソッドのそれぞれが，有限ステップで完了するということである．

ウェイトフリー性は一番望ましい性質であるが，パフォーマンスに劣る場合がある [5, chapter 3]．また，オブストラクションフリー性はライブロックが起きないことを保証しない．そのため，この 3 つの性質のうち保証する性質やパフォーマンスの観点からロックフリー性が望ましいと考えられる [5, chapter 3]．

### 1.2　本研究の目的と貢献

現在まで様々なロックフリーアルゴリズム [4, 13, 14, 18] やロックフリー性を形式的 (formal) に保証する方法 [3, 6, 7, 9, 12, 19] が研究されてきた．既存研究において，ロックフリー性を証明する方法として，自然言語による非形式的な証明や，紙の上 [3, 6] ないし定理証明支援器上の形式的な証明 [7, 19] がある．また上記の方法以外にも，軽量な形式手法とされる型システムに基づく手法 [9, 12] がある．定理証明支援器（のみ）を用いる方法や，可能な遷移をすべて具体的に列挙する方法に比べ，型システムを用いる方法は，特に型推論により型注釈を省略できる場合，プログラマの負担を軽減することができると期待される．

本研究の目的は，Kobayashi と Sangiorgi のハイブリッド型システム [12] と Kobayashi の静的解析器 TyPiCal [8] を用いて，Treiber のロックフリースタック [18] のロックフリー性を証明することである．ハイブリッド型システムを用いてロックフリー性を検証するためには，「ロバストな」（すなわち，外部と通信した際の）停止性・デッドロックフリー性・合流性を保証すれば十分である [12, section 3 および theorem 4.2]．そこでまずロバストなデッドロックフリー性を保証するために，TyPiCal を用いて，ロックフリースタックのデッドロックフリー性を検証することにした．具体的には，再帰型を持つ連結リスト構造を用いてロックフリースタックを表現し，TyPiCal で検証を試みた．その際，TyPiCal では再帰型を持つデータ構造に対する操作はプリミティブ関数として定義する必要があり，それにともない新たに発生した制約の生成と解消を実装した．ロバストな停止性や合流性はシステムの一部の遷移を列挙して検証を試み，ロックフリー性の定義の違いにより生じる問題点について考察する．

## 2　準備

### 2.1　Treiber のロックフリースタック

Treiber のロックフリースタック [18] は，図 1 の擬似コードで表せる．スタックは連結リストとして実装され，`Node` 構造体はリストのセルを表している．グローバル変数 `S` の値はスタックの先頭を指すポインタであり，初期値として空のスタックを表す `NULL` が代入されている．

**`push`** 関数は，引数 `v` の値をスタックの先頭に追加する関数である．この関数ではまず，新しいセル `x` を作り，`v` を格納する．次に，スタックの先頭を指すポインタを変数 `S` から読み出し，変数 `t` に格納する．14 行目から 20 行目までは CAS (compare-and-swap) 命令を表す **`atomic`** ブロックであり [6]，変数 `S` から再び読み出した値を変数 `s` に格納し，`t` と `s` の値が等しいかどうか確認する．等しい場合，`x` の値を `S` に書き込み，処理を終了する．等しくない場合，ループの先頭である 12 行目に戻り，上の処理をやり直す．

**`pop`** 関数は，スタックの先頭要素を取り出す関数である．スタックが空である場合，デフォルト値として `EMPTY` を返す．**`pop`** は **`push`** と同じように，最初に変数 `S` から読み出した値を `t` に格納する．そして，次のセルを指すポインタを変数 `x` に格納する．33 行目から 39 行目までは **`push`** 関数と同様に，変数 `S` から再び読み出した値が `t` の値と等しければ `x` の値を `S` に書き込み，そうでなければループの先頭に戻る．

**`push`** 関数や **`pop`** 関数がロックフリーである直観的理由は以下である．**`push`** や **`pop`** の成否は，CAS 命令の成否に対応し，CAS 命令が成功した場合は **`while`** ループが終了し，操作が完了する．逆に CAS 命令が失敗した場合は，スタックが変更されている，すなわち他のスレッドの **`push`** や **`pop`** の処理が成功したことを意味する．よって，少なくとも一つの **`push`** または **`pop`** 操作は完了する．以上はどのようなスケジューリングの下でも成り立つ．したがって，このスタックの実装はロックフリーであることがわかる．

```
1  struct Node {
2    int data;
3    Node *next;
4  };
5
6  Node *S = NULL;
7
8  void push(int v) {
9    Node *x = new Node();
10   x->data = v;
11   while (true) {
12     Node *t = S;
13     x->next = t;
14     atomic { // CAS(&S, t, x)
15       Node *s = S;
16       if (s == t) {
17         S = x;
18         break;
19       }
20     }
21   }
22 }
23
24 int pop() {
25   int ret = 0;
26   while (true) {
27     Node *t = S;
28     if (t == NULL) {
29       return EMPTY;
30     } else {
31       Node *x = t->next;
32       ret = t->data;
33       atomic { // CAS(&S, t, x)
34         Node *s = S;
35         if (t == s) {
36           S = x;
37           return ret;
38         }
39       }
40     }
41   }
42 }
```

図 1　ロックフリースタックの擬似コード

### 2. 2　TyPiCal の対象言語：$\pi$ 計算

TyPiCal [8] の対象言語である $\pi$ 計算について説明する．$\pi$ 計算とは，並行計算のモデルの一つである [15,16]．本論文では TyPiCal の実装 [8]，ハイブリッド型システム [12]，デッドロックフリー性を保証する型システム [10] のそれぞれの対象言語を合わせたような $\pi$ 計算を考える．

$\pi$ 計算では並行システムはプロセスとして表され，プロセスの集合は図 2. 2 のように定義される．ここでメタ変数 $a$ はチャンネル，$x$ や $y$ は変数を表す．メタ変数 $v$ や $w$ はチャンネルや変数，定数を含め，値を表す．プロセス $\mathbf{0}$ は何もしないプロセスを表す．プロセス $v!^{\chi}w.\,P$ はチャンネル $v$（値 $v$ が変数 $x$ の場合は，その変数 $x$ に束縛されたチャンネル．以下同様）に $w$ を送信した後，$P$ を実行する．プロセス $v?^{\chi}y.\,P$ はチャンネル $v$ から受信した値を $y$ に束縛した後，$P$ を実行する．アノテーション $\chi$ は $\circ$ か $\bullet$ であり，アノテーションがついている送受信が成功してほしい（$\circ$）か，そうでないか（$\bullet$）というプログラマの意図を表す．アノテーション $\chi$ が $\circ$ であるとき，プレフィックス $v!^{\chi}w$ ないし $v?^{\chi}y$ が**マークつき** (marked) であるという．プロセス $(P \mid Q)$ は $P$ と $Q$ を並列に実行する．プロセス $*P$ は無限個の $P$ の並列実行（複製）を表す．プロセス $(\nu a)P$ は $P$ において $a$ がフレッシュであることを表す．プロセス $\mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q$ は $v$ が $\mathit{true}$ のとき $P$，$\mathit{false}$ のとき $Q$ を実行する．プロセス $\mathbf{let}\ x = e\ \mathbf{in}\ P$ は，式 $e$ を評価した値を $x$ に束縛した後，$P$ を実行する．本論文では [8,10] と同様に，**let** を用いて式の評価を明示し，送受信 !, ? の対象および **if** の条件部分は値（変数の場合を含む）に限定している．ただし，実際にプロセスの例を書く際は，しばしば **let** を展開した略記を用いる．

ラベルつき遷移関係 $P \xrightarrow{\eta} Q$ は，プロセス $P$ が，ラベル $\eta$ が表す動作を行なってプロセス $Q$ に遷移することを表す．例えば，プロセス $P$ がそのプロセス内部で送受信を行い，プロセス $Q$ に遷移するとき，ラベル $\tau$ を用いて，$P \xrightarrow{\tau} Q$ と書く．ラベルつき遷移関係 $\xrightarrow{\tau}$ の反射的推移的閉包を $\xrightarrow{\tau}{}^{*}$ と書く．また，プロセス $P$ に関して，$P \xrightarrow{\eta} Q$ や $P \xrightarrow{\tau}{}^{*} Q$ となるような $Q$ が存在するとき，それぞれ $P \xrightarrow{\eta}$ や $P \xrightarrow{\tau}{}^{*}$

$$
\begin{array}{rcll}
P \text{ (processes)} & ::= & \mathbf{0} & \text{(inaction)} \\
& | & v!^{\chi}w.\ P & \text{(output)} \\
& | & v?^{\chi}y.\ P & \text{(input)} \\
& | & (P \mid Q) & \text{(parallel composition)} \\
& | & *P & \text{(replication)} \\
& | & (\nu a)P & \text{(restriction)} \\
& | & \textbf{if } v \textbf{ then } P \textbf{ else } Q & \text{(conditional)} \\
& | & \textbf{let } x = e \textbf{ in } P & \text{(evaluation)} \\
e \text{ (expressions)} & ::= & \mathit{true} \mid \mathit{false} \mid x \mid a \mid \langle e_1, e_2 \rangle \mid \mathit{proj}_1(e) \mid \mathit{proj}_2(e) & \\
v, w \text{ (values)} & ::= & \mathit{true} \mid \mathit{false} \mid x \mid a \mid \langle v_1, v_2 \rangle &
\end{array}
$$

**図 2　TyPiCal の対象言語：$\pi$ 計算**

と書く．ラベルつき遷移関係の定義は [10–12] などを参照されたい．

### 2.3　ロックフリー性の定義

プレフィックスがトップレベルにあるとは，その送受信が他の送受信や複製の下にないことをいう（複製の下のプレフィックスは遷移が進めば複製の外に出てくるので，議論を簡単にするために省いている）[12, p. 8].

**定義 1 (デッドロックフリー性 [12, Definition 2.6])** プロセス $P$ がデッドロックフリーであるとは，$P \xrightarrow{\tau}{}^{*} Q$ なるすべての $Q$ に対して，マークつきプレフィックスが少なくとも一つ $Q$ のトップレベルにあるとき，$Q \xrightarrow{\tau}$ が成り立つことをいう． □

ロックフリー性を定義する際，一つ一つのマークつきプレフィックスの成否を追跡する必要がある．プロセス $P$ 中のトップレベルのプレフィックスのアノテーション $\circ$ をどれか一つだけ □ に書き換えたプロセスの集合を，$P$ の**タグづけ** (tagging) という．また，送信と受信の少なくとも一方に □ がついているような $\tau$ 遷移を $\xrightarrow{\tau^{\Box}}$ と書く．

**定義 2 (強いロックフリー性 [12, Definition 2.8])** プロセス $P$ が強いロックフリー性を満たすとは，$P \xrightarrow{\tau}{}^{*} Q$ なるすべての $Q$ に対して，$Q$ のタグづけに属するそれぞれのプロセスの遷移列のうち，full かつ strongly fair であるすべての遷移列が，遷移 $\xrightarrow{\tau^{\Box}}$ を含んでいることをいう． □

ここで遷移列が full であるとは，遷移列が無限であるか，これ以上遷移できないプロセスで終わることをいう．また，遷移列が strongly fair であるとは，直観的には，同一のトップレベルの入出力プレフィックスの組が無限回出現しないことをいう [12, p. 9].

デッドロックフリー性と強いロックフリー性の違いは，ライブロックの有無である．未完了の送受信がある場合，デッドロックフリー性はプロセスが更に $\tau$ 遷移することのみを保証し，ロックフリー性はその特定の送受信が有限回の $\tau$ 遷移で成功することを保証する．

1.1 節で示した [5] でのロックフリー性の定義と，本節で示した [12] でのロックフリー性の定義は異なっている．[5] の定義は，どのようなスケジューリングにおいても，ある少なくとも一つの操作が完了することを意味しているが，[12] の定義は，strongly fair なスケジューリングの下で，すべての操作がいずれ完了することを意味している．したがって，無限個の操作（例えば `push`(1)，`push`(2)，...）を並行に行う場合，前者では（たとえ strongly fair なスケジューリングの下でも）一部の操作（例えば `push`(奇数) のみ）しか成功しない可能性がある一方，後者では（strongly fair なスケジューリングを仮定すれば）すべての操作がいずれ成功する．

$$\frac{\Delta \models_{\texttt{RD}} P \qquad \mathit{Erase}(\Delta) \models_{\texttt{RTer}} P \qquad \mathit{Erase}(\Delta) \models_{\texttt{RConf}} P \qquad \mathit{nocap}(\Delta)}{\Delta \vdash_{\texttt{SLT}} P}\,(\text{SLT-Hyb})$$

**図 3　ハイブリッド型システムのための型つけ規則**

### 2.4　デッドロックフリー性を保証する型システム

本研究が用いるデッドロックフリー性を保証する型システム [10, 11, 17] は，$\pi$ 計算の単純型システムに，usage と obligation level，capability level を導入して拡張した型システムである．型の集合は以下のように定義される．

$$\tau \text{ (types)} ::= \mathbf{bool} \mid \tau_1 \times \tau_2 \mid \mathbf{chan}(\tau, U)$$

チャンネル型 $\mathbf{chan}(\tau, U)$ は，$\tau$ 型の値を usage $U$ に従って送受信するチャンネルの型を表す．

usage とはチャンネルがどのように送受信に使われるかを表した式であり，以下のような抽象構文で定義される．

$$U ::= \mathbf{0} \mid \alpha^{t_o}_{t_c}.\,U \mid (U_1 \mid U_2) \mid \ldots$$
$$\alpha ::= \,!\, \mid\, ?$$

ここで，! は送信を表し，? は受信を表す．また，$t_o$ は obligation level，$t_c$ は capability level を表し，それぞれ自然数か $\infty$ である．obligation level と capability level は，直観的には時刻に対応する．$\alpha^{t_o}_{t_c}$ の obligation level $t_o$ は，時刻 $t_o$ までに，その送信または受信を行わなければならないことを表す．ただし，$t_o = \infty$ のときは，送受信を行わなくてもよいことを表す．また，capability level $t_c$ は，その送信または受信を行なったら時刻 $t_c$ までに成功することを表す．ただし，$t_c = \infty$ のときは，成功することを保証しない．

あるプロセスがデッドロックフリーであることを保証するためには，各チャンネルがデッドロックしないように使われることを，以下のように usage を通して確認すればよい．

まず，ある送信（または受信）が成功するためには，それ以前の時刻に，対応する受信（または送信）が行われる必要がある．例えば，プロセス $x!0 \mid x?y$ における $x$ の usage $!^{t_0}_{t_1} \mid ?^{t_2}_{t_3}$ では，capability level $t_1$ の送信 ! に対応する受信 ? の obligation level $t_2$ は，$t_1$ 以下である必要がある．なぜなら直観的には，送信が時刻 $t_1$ までに成功するためには，対応する受信が時刻 $t_1$ 以前に行われている必要があるからである．

また，ある送受信の obligation $t_o$ が他の送受信の capability $t_c$ で遅延されている場合，$t_o > t_c$ という順序関係が生じる．例えば，プロセス $x?z.\,y!0 \mid y?z'.\,x!0$ において，$x$ の usage を $?^{t_0}_{t_1} \mid !^{t_2}_{t_3}$ （ただし前述の制約より $t_0 \leq t_3$ かつ $t_2 \leq t_1$），$y$ の usage を $?^{t_4}_{t_5} \mid !^{t_6}_{t_7}$ （ただし $t_4 \leq t_7$ かつ $t_6 \leq t_5$）とする．$y!0$ は $x?z$ のあとにあるため，$t_1 < t_6$ という制約が生じる．また，$x!0$ が $y?z'$ のあとにあるため，$t_5 < t_2$ という制約が生じる．これらの制約から，$t_1 = t_6 = t_5 = t_2 = \infty$ となる（ただし $\infty < \infty$ は成り立つとする）．つまり，受信 $x?z$ や $y?z'$ は成功しない可能性があることがわかる．

さらに，[10] の型システムでは，usage と obligation level，capability level に加えて，チャンネルが作成された順序を表す関係 $\prec$ を導入している．型環境 $\Gamma$ とチャンネルの順序関係 $\prec$ のもとでプロセス $P$ に型がつくとき，$P$ はデッドロックフリーである．詳細は [10, section 3] を参照されたい．

### 2.5　ハイブリッド型システム

強いロックフリー性を保証するハイブリッド型システム [12] は，強いロックフリー性を保証するハイブリッドでない（完全に静的な）型システム [9] に，図 3 の型つけ規則を追加した型システムである．型環境 $\Delta$ はプロセス $P$ に自由出現する名前に関する仮定を表す（*Erase* は型の中の usage を消去する）．前提 $\Delta \models_{\texttt{RD}} P$，$\mathit{Erase}(\Delta) \models_{\texttt{RTer}} P$，$\mathit{Erase}(\Delta) \models_{\texttt{RConf}} P$ は，それぞれロバストなデッドロックフリー性，ロバストな停止性，ロバストな合流性を表す．ロバストな性質（停止性，デッドロックフリー性，合流性のいずれか）とは，大まかには，任意のプロセス $Q$ と並列に実行しても，$P$ がその性質を満たすことを意味する．$Q$ は任意のため，$\Delta$ はいかなる capability も含まないとする（$\mathit{nocap}(\Delta)$）．ただし，5 節で検証するロックフリースタックのシステム全体は自由な名前を

```
1 *makeCell?r.
2   new readData in new readNext in new writeData in new writeNext in
3   let quad = makeQuadruple (readData, readNext, writeData, writeNext) in
4   r!quad | new data in new next in data!0 | next!null
5     | *readData?r. data?v. (r!v | data!v)
6     | *readNext?r. next?v. (r!v | next!v)
7     | *writeData?(v2,r). data?v1. (r!0 | data!v2)
8     | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
```

図 4 makeCell

含まず，型環境も空としてよい．そのような場合，ロバストな性質は通常の性質と同様である．

ここでいう合流性とは概ね以下の意味である [12, Definition 3.8]．プロセス $P$ が部分的な合流性を持つとは，$P_1 \xleftarrow{\tau,S_1} P \xrightarrow{\eta,S_2} P_2$ ならば常に，$\eta = \tau \wedge S_1 = S_2$ であるか，$P_1 \xrightarrow{\eta,S_1}\equiv\xleftarrow{\tau,S_2} P_2$ であることをいう．ここで，$S_1$，$S_2$ はその遷移に関わったチャンネルの集合を表す [12, p. 18]．プロセス $P$ がロバストな合流性を持つとは，任意の閉じた代入 $\sigma$ について，$\sigma P \xrightarrow{\eta_1} \cdots \xrightarrow{\eta_k} Q$ ならば常に，$Q$ が部分的な合流性を持つことをいう．

規則 (SLT-Hyb) は，上述の仮定が成り立てば，プロセス $P$ はロックフリーであり，$P$ が（他の必ずしも停止しないプロセスに対して）strongly fair にスケジュールされれば，$\Delta$ に含まれる obligation を果たす，ということを主張している．この規則の正しさ（健全性）は，直観的には以下のように理解できる．まず，ロバストな停止性より，プロセス $P$ の遷移列は有限である．すると，ロバストなデッドロックフリー性 [12, Definition 3.7] より，$P$ は少なくとも 1 回は obligation を果たせる状態になる．よって，ロバストな合流性から，$P$ は必ず obligation を果たす．

## 3 ロックフリースタックの $\pi$ 計算での表現

2.1 節で紹介したロックフリースタックを $\pi$ 計算で表現するためには，まず，Node 構造体のようなセルをプロセスで表現する必要がある．セルを表現するプロセスには，セルが保持する値と次のセルのポインタを読み書きするために，4 つのチャンネルを用意する．

図 4 のプロセス*makeCell?r...は返信用のチャンネル r を受信し，上述の 4 つのチャンネルを組として r に返信するサーバである．[†1] このサーバはチャンネル r を受信した後，まず readData，readNext，writeData，writeNext の 4 つのフレッシュなチャンネルを作成する．次に，プリミティブ関数 makeQuadruple を用い，この 4 つのチャンネルを組にして，チャンネル r に送信する．チャンネル data と next は，セルの要素と，次のセルを指すポインタをそれぞれ保持するチャンネルである．*readData?r...と*readNext?r...は，返信用のチャンネル r を渡されたら，それぞれ data または next から値を取り出して，r に渡すとともに，再び data または next に戻す．*writeData?(v2,r)...と*writeNext?(v2,r)...は，それぞれ data または next に書き込む値と，処理が完了したことを通知するチャンネルの組を受け取る．そして data または next から値を取り出し，r に通知を行うとともに，data または next に新しい値 v2 を渡す．

図 5 は push 関数を $\pi$ 計算で表現したプロセスである．このサーバは全体的動作としては，スタックに追加する値 value と，処理が完了したことを知らせるためのチャンネル reply を受け取って，スタックに値 value を追加した後，reply に 0 を送信する．詳細としては，まず 2 行目で makeCell に送信したチャンネルから 4 つ組 quad を受け取る．それから 5 行目でセルに value を格納する．6 行目以降は図 1 の擬似コードの while ループに対応する．擬似コード

†1 本論文ではしばしば $\pi$ 計算のプロセスを，TyPiCal の実装に近い具体文法を用いて表す．

```
1  *push?(value, reply).
2      new ref in makeCell!ref. ref?quad.
3      let writeData = third quad in
4      let writeNext = fourth quad in
5      new a in writeData!(value, a). a?l.
6      new while in while!reply
7      | *while?r. stack?t. ( stack!t
8        | new a in writeNext!(t, a). a?l. lock?l.
9          stack?s. ( stack!s
10         | if s = t
11           then stack?u. ( stack!quad | lock!0 | r!0 )
12           else ( lock!0 | while!r )))
```

**図 5** push

```
1  *pop?reply.
2     new while in while!reply
3     | *while?r. stack?t. ( stack!t
4       | if s = null
5         then r!0
6         else let readData = first t in
7              let readNext = second t in
8              new a in readData!a. a?v.
9              new b in readNext!b. b?x.
10             lock?l. stack?s. ( stack!s
11             | if s = t
12               then stack?u. ( stack!x | lock!0 | r!v )
13               else ( lock!0 | while!r )))
```

**図 6** pop

における atomic ブロックはグローバルなチャンネル lock に値を送受信することで実現し，while ループは再帰的なプロセスで表している．返信用のチャンネル reply をチャンネル while に渡している理由は，デッドロックフリー性を保証する型システムにおける reply への送信の obligation を，ループを繰り返すたびに受け渡しするためである．

図 6 は，ロックフリースタックの pop 関数を $\pi$ 計算で表現したプロセスである．このプロセスは，チャンネル pop から返信用のチャンネル reply を受け取り，スタックの先頭の要素を reply に送信する．スタックが空（null）である場合は，デフォルトの値として 0 を送信する．全体の流れは，push 関数の表現と同様である．

ロックフリースタックに push や pop を無限回行うクライアント側のプロセスは図 7 のようになる．pop では，返信用のチャンネル r を pop に送信し，r から v を受信する．push では，スタックに追加する値と，処理が完了したことを伝えるチャンネル r を組にして送信し，r から受信を行う．

上述のロックフリースタックのサーバとクライアントの $\pi$ 計算での表現は，1.1 節の意味ではロックフリーだが，定義 2 の意味での強いロックフリー性は満たさない．なぜならば，無限個のクライアントのう

```
*(new r in pop!r. r?rep) | *(new r in push!(1,r). r?rep)
```

**図 7**　無限個のクライアント

$$
\begin{array}{rcl}
P_0 & \triangleq & (\nu r)(\mathtt{push}!\langle 1, r\rangle.\ r?rep) \\
Q_0 & \triangleq & (\nu r)(\mathtt{pop}!r.\ r?rep) \\
\mathit{System} & \triangleq & (\nu \mathtt{makeQuadruple}, \mathtt{push}, \mathtt{pop}, \mathtt{stack}, \mathtt{lock}) \\
 & & (\mathtt{stack!null} \mid \mathtt{lock!0} \mid *\mathit{Push} \mid *\mathit{Pop} \mid P_0 \mid Q_0)
\end{array}
$$

**図 8**　ロックフリースタックのシステム全体（push と pop が **1** 回ずつの場合）

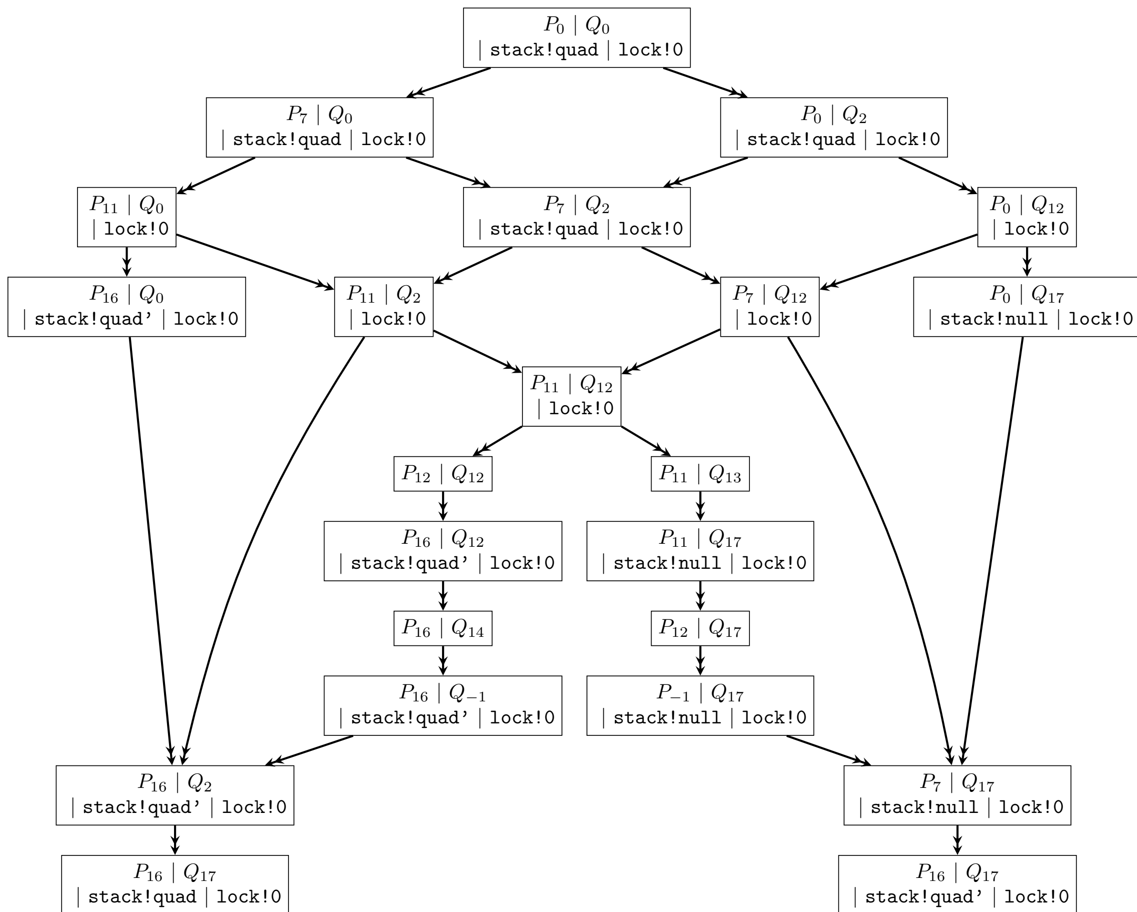


**図 9**　クライアントの状態遷移

ち，一部（例えば「奇数番」）のクライアントのみがそれぞれ有限時間で操作を完了し，他（例えば「偶数番」）のクライアントは永久に操作を完了しないような，full かつ strongly fair な遷移列が存在するためである．一方，定義 2 を弱め，すべての $Q$ について，$\xrightarrow{\tau^{\Box}}$ を含む遷移列が存在するという**弱いロックフリー性**は満たされる．

push 操作や pop 操作が有限回のシステムであれば，

$$\begin{aligned}
\texttt{makeQuadruple}: \quad & \mathbf{chan}(\mathbf{chan}(\mathbf{int}, !), *!) \times \\
& \mathbf{chan}(\mathbf{chan}(\mathbf{quad_t}, !), *!) \times \\
& \mathbf{chan}(\mathbf{int} \times \mathbf{chan}(\mathbf{int}, !), *!) \times \\
& \mathbf{chan}(\mathbf{quad_t} \times \mathbf{chan}(\mathbf{int}, 1), *!) \\
& \longrightarrow \mathbf{quad_t} \\
\texttt{first}: \quad & \mathbf{quad_t} \longrightarrow \mathbf{chan}(\mathbf{chan}(\mathbf{int}, !), *!) \\
\texttt{second}: \quad & \mathbf{quad_t} \longrightarrow \mathbf{chan}(\mathbf{chan}(\mathbf{quad_t}, !), *!) \\
\texttt{third}: \quad & \mathbf{quad_t} \longrightarrow \mathbf{chan}(\mathbf{int} \times \mathbf{chan}(\mathbf{int}, !), *!) \\
\texttt{fourth}: \quad & \mathbf{quad_t} \longrightarrow \mathbf{chan}(\mathbf{quad_t} \times \mathbf{chan}(\mathbf{int}, !), *!)
\end{aligned}$$

**図 10　プリミティブ関数の型**

強いロックフリー性を満たす．例えば push と pop をそれぞれ 1 回ずつ並行に行うロックフリースタックのシステム全体は図 8 のように表される．ここで $Push$ と $Pop$ はそれぞれ，図 5 の push サーバと，図 6 の pop サーバを表している．また，$P$ と $Q$ はそれぞれ，push を行うクライアントと，pop を行うクライアントを表している．

後者のシステム全体の状態遷移を列挙する．stack に何らかの quad が出力されている状態から，push と pop をそれぞれ 1 回ずつ並行に行うクライアント $P_0 \mid Q_0$（および，そのときの stack と lock への出力プロセス）の状態遷移図を図 9 に示す（それら以外のプロセスは変化しないので省略する）．この図では 1 ステップ以上の遷移を $\twoheadrightarrow$ で表している．push を行うクライアントの状態を $P_i$ $(i = 0, 1, \ldots, 16, -1)$ で表し，pop を行うクライアントの状態を $Q_j$ $(j = 0, 1, \ldots, 17, -1)$ で表すことにする．$P_i$ および $Q_j$ の具体的な定義はそれぞれ付録 A と B に示す．push と pop を行うクライアントが並行に実行されている状態 $P_i \mid Q_j$ を考えると，$i$ と $j$ の値の組が取りうる範囲とその理由は以下のとおりである．

$0 \le i \le 16 \wedge 0 \le j \le 2$：　push 処理が任意に進み（$0 \le i \le 16$），pop 処理は 1 回目の stack?t 以前まで進んでいる（$0 \le j \le 2$）．

$0 \le i \le 7 \wedge 0 \le j \le 17$：　pop 処理が任意に進み（$0 \le j \le 17$），push 処理は 1 回目の stack?t 以前まで進んでいる （$0 \le i \le 7$）．

$i = 16 \wedge 0 \le j \le 17$：　push が完了した後（$i = 16$），pop の処理が任意に進んでいる（$0 \le j \le 17$）．

$0 \le i \le 16 \wedge j = 17$：　pop が完了した後（$j = 17$），push の処理が任意に進んでいる（$0 \le i \le 16$）．

$7 \le i \le 11 \wedge 2 \le j \le 12$：　push と pop が 1 回目の stack?t を実行後，lock?l をとる以前まで進んでいる．

$11 \le i \le 16 \wedge j = 12$：　push が先に lock?l でロックを取得し（$i = 12$），push が完了する（$i = 16$）．

$i = 16 \wedge j = 12, 13, 14, -1$：　push が完了した後（$i = 16$），pop がロックを取得し（$j = 13$），スタックの先頭が変わっているため，競合を検出して再試行を行う（$j = 2$）．

$i = 11 \wedge 12 \le j \le 17$：　pop が先に lock?l でロックを取得し（$j = 13$），pop が完了する（$j = 17$）．

$i = 11, 12, 13, -1 \wedge j = 17$：　pop が完了した後（$j = 17$），push がロックを取得し（$i = 12$），スタックの先頭が変わっているため，競合を検出して再試行を行う（$i = 7$）．

## 4　TyPiCal の拡張とデッドロックフリー性の検証

### 4.1　再帰型を持つ 4 つ組プリミティブの追加

本節では，3 節で述べた $\pi$ 計算におけるロックフリースタックの表現のデッドロックフリー性を TyPiCal で検証する．図 4 の 4 つ組 quad で表現されるセルは，次のセルへのポインタ（writeNext の第 1 引数 v2 および readNext の返り値）を持っているため，再帰型を必要とする．TyPiCal ではユーザは新たな再帰型を定義することができないため，TyPiCal の

実装に新たな再帰型と，それを用いるプリミティブ関数を追加した．具体的には，再帰型を持つ 4 つ組を作る関数 `makeQuadruple` と，その要素を取り出す関数 `first`, `second`, `third`, `fourth` を追加した．

4 つ組 quad の型 **quad_t** の定義は，チャンネルの usage の部分を除くと次のとおりである．

$$
\begin{aligned}
\mathbf{quad_t} \triangleq \mu\alpha.\; & \\
& \mathbf{chan}(\mathbf{chan}(\mathbf{int})) && // \texttt{readData} \\
\times\; & \mathbf{chan}(\mathbf{chan}(\alpha)) && // \texttt{readNext} \\
\times\; & \mathbf{chan}(\mathbf{int} \times \mathbf{chan}(\mathbf{int})) && // \texttt{writeData} \\
\times\; & \mathbf{chan}(\alpha \times \mathbf{chan}(\mathbf{int})) && // \texttt{writeNext}
\end{aligned}
$$

4 つ組のそれぞれのチャンネルの usage を含む型を考えると以下のようになる．

$$
\begin{aligned}
\texttt{readData} &: \mathbf{chan}(\mathbf{chan}(\mathbf{int}, !), *!) \\
\texttt{readNext} &: \mathbf{chan}(\mathbf{chan}(\mathbf{quad_t}, !), *!) \\
\texttt{writeData} &: \mathbf{chan}(\mathbf{int} \times \mathbf{chan}(\mathbf{int}, !), *!) \\
\texttt{writeNext} &: \mathbf{chan}(\mathbf{quad_t} \times \mathbf{chan}(\mathbf{int}, !), *!)
\end{aligned}
$$

以上を踏まえると，追加するプリミティブ関数の型は図 10 のようになる．

これらのプリミティブ関数を追加した結果，新たな形の制約が発生したため，それを解消する方法を実装した．具体的には，subusage 制約 $U_1 \leq U_2$ において，従前の TyPiCal では $U_1$ は usage 変数 $\alpha$ に限られていたが，今回の拡張により新たに $U_1$ が $!^{t_o}_{t_c}.\,\mathbf{0}$, $*!^{t_o}_{t_c}.\,\mathbf{0}$, $\uparrow^t (!^{t_o}_{t_c}.\,\mathbf{0})$, $\uparrow^t (*!^{t_o}_{t_c}.\,\mathbf{0})$ のケースも発生したため，本研究で必要となった，$U_2$ に ? が出現しないケースについて，subusage の定義に従い制約の解消を実装した．具体的には，条件 $cap_!(U_1) \leq cap_!(U_2)$ [10, Definition 35, 3.]（他の条件は自明）から，capability level に関する制約（それら自体は既存）を生成する関数を実装した．

### 4.2 TyPiCal での検証結果

TyPiCal に，無限回の push および pop 操作を並行に行うロックフリースタックのプロセスを入力した結果の出力を図 11 に示す．`!!`や`??`は，その送受信の成功が保証されている（マークつきである）ことを表す．pop 操作を行うクライアント側のプロセス（7 行目）において，処理が終わった通知を必ず受け取れることが保証されている（`x??rep`）．つまり，pop サーバはデッドロックフリーであることがわかる．8 行目の push 操作についても同様である．

push および pop 操作が有限回の場合も同様である．

```
1 new makeCell in new push in new pop in
2 new stack in new lock in
3 stack!null | lock!0
4 | (*makeCell??r . ...)
5 | (*push?r. ...)
6 | (*pop?r. ...)
7 | *(new r in pop!!r . r??rep)
8 | *(new r in push!!(1,r) . r??rep)
```

図 11　TyPiCal での検証結果の一部

## 5 ロックフリー性の検証

本節では，まず図 8 の，有限回の操作に対するロックフリースタックのロックフリー性を考える．図 8 のシステムは，4.2 節のとおりデッドロックフリーであり，さらに図 9 のとおり停止性も成り立つ（デッドロックフリー性も図 9 から確認することもできる）．したがって，図 8 のシステムはロックフリーであるといえる．

一方，図 7 の，無限回の操作を行うロックフリースタックは，状態数が無限になるため，図 9 のようにすべての遷移を列挙することは容易でない．そこで，push サーバおよび pop サーバは図 3 のハイブリッド型システムで追加された型つけ規則を用い，サーバ以外の部分は従来のロックフリー性を保証する完全に静的な型システム [9] を用いて型つけすることを考える（ [12, Example 3.12] と同様）．

図 3 の規則では，ロバストなデッドロックフリー性，ロバストな合流性，ロバストな停止性を示す必要がある．まず，4.2 節のように，TyPiCal が実装しているデッドロックフリー性を保証する型システムで型つけが成功すれば，ロバストなデッドロックフリー性も成り立つことが知られている [12, Theorem 3.9]．（サーバの）ロバストな停止性は，外部との送受信は有限回である前提なので [12, Definition 3.6]，スタック操作が有限個の場合と同様に成り立つ．

問題は（サーバの）ロバストな合流性である．ハイ

ブリッド型システムでいう合流性は 1 ステップでの合流性という定義 [12, Definition 3.8] なので，そのままでは成り立たない．1 ステップでの合流性を要求する理由は，ある遷移列で obligation が果たされれば，同じ obligation がすべての遷移列で果たされることを示す際，前者に含まれる遷移は，後者にも含まれることを示すためである [12, Lemma C.1]．1 ステップではなく複数ステップでの合流性に弱めることも可能だが，fairness の前提を適用するため，同じ遷移が先頭に来る（enable されている）必要がある．

そのように合流性の条件を弱めても，本論文のようなロックフリースタックのサーバでは，例えば `lock` や `stack` の受信後，直ちに送信は行えないので，成り立たない．実際，無限個の `push` 操作や `pop` 操作を並行に行う場合，`push` のみが実行され続けて，`pop` が永久に実行されない可能性があるので，ロックフリー性が成り立たない．これは 2.3 節すなわち [12, p. 9] でいう strongly fair な遷移を前提にしても，`push` のみが実行されている場合，図 6 の 3 行目の `stack?t` に対応する図 5 の 3 行目の `stack!t` および 10 行目の `stack!s` が，`push` 操作のたびに複製されて異なる出現とみなされるため，`pop` 操作の成功が保証されないからである．[12, p. 9] のような fairness の定義を変更し，送受信の組ではなく `stack?t` のような受信（ないし送信）のみに注目して成功を保証するようなスケジューリングの定義は今後の課題である．そのように定義を変更すれば，前述の obligation がすべての遷移列で果たされることや，ひいては `push` サーバや `pop` サーバのロックフリー性も証明できる可能性がある．

$\pi$ 計算でのスタックの表現を [12, Example 3.12] の二分探索木のように根本的に変更すれば，ハイブリッド型システムでロックフリー性を保証することも可能である．本研究では，共有メモリを用いた Treiber のロックフリースタックを $\pi$ 計算で表現してロックフリー性を検証することを試みた．

2.3 節で述べたとおり，1.1 節でいうアルゴリズムのロックフリー性は，2.3 節のプロセスのロックフリー性 [12, Definition 2.8] と異なり，fair なスケジューリングを仮定しない一方，少なくとも 1 つの操作が完了することのみを主張している．無限個の `push` 操作や `pop` 操作がある場合（あるいは無限にスレッドを生成できるシステムの場合），fair なスケジューリングを仮定しなければ，すべてのスレッドが途中までしか実行されず，いずれの操作も永久に完了しない可能性があるので，後者のロックフリー性は明らかに成り立たない．

## 6 関連研究

すでに述べたロックフリーアルゴリズム [4,13,14,18] や，ロックフリー性を保証する型システム [9,12] 以外に以下のような研究がある．

Hoffmann ら [6] はロックフリー性の定義を形式化して，それを検証するための分離論理の拡張を提案し，Treiber のロックフリースタックのロックフリー性を紙の上で形式的に証明した．

Jia ら [7] は，“read, compute and update” という形をしたアルゴリズムのソースコードに補助変数を割り当てることによりロックフリー性を検証する手法を提案した．具体的には，共有データへの書き込みが成功した場合には補助変数の値をインクリメントし，失敗した場合には，他のスレッドにより補助変数の値がインクリメントされている（他のスレッドが書き込みに成功している）ことを確認する．これにより，スレッドが 1 つずつ減っていくことや，プログラム全体の停止性を議論している．また，その手法を自動証明ツール Cave として実装し，ロックフリーなスタックやキューのロックフリー性を証明した．

Pinto [1] は，Total-TaDA という，ノンブロッキングなプログラムの停止性を証明する分離論理の拡張を提案し，それを用いてロックフリーなスタックやキューの紙の上での証明を行なった．

佐久間 [19] は，Hoffmann らの分離論理の拡張に基づき，Treiber のスタックの `push` のロックフリー性を，Coq を用いて検証した．

これらと比較すると本研究は，既存の型システムを用いた部分については，分離論理の拡張に基づく証明より簡便な検証が可能であった．

## 7 結論と今後の課題

本研究では，$\pi$ 計算のプロセスのロックフリー性を型システムにもとづき検証するツールである TyPiCal を，再帰型を持つセル構造で拡張して Treiber のロックフリースタックを表現し，そのデッドロックフリー性を検証した．デッドロックに限らないロックフリー性は，有限回のスタック操作についてはハイブリッド型システム [12] を用いて検証した．無限回の並行なスタック操作の場合は，そもそもロックフリー性が成り立たない一方，fairness の仮定を変更すれば証明できる可能性を示唆した．また，ハイブリッド型システムでロックフリー性を検証する際に必要な停止性は，本研究ではすべての遷移列を列挙して検証したが，ツールによる自動化や型システム [2] の利用も考えられる．

**謝辞** TyPiCal やハイブリッド型システムに関する質問にご回答くださった小林直樹氏に深謝します．本研究は JSPS 科研費 JP15H02681，JP20H04161 の助成を受けたものです．

### 参 考 文 献

[1] da Rocha Pinto, P.: *Reasoning with time and data abstractions*, PhD Thesis, Imperial College London, UK, 2016.

[2] Deng, Y. and Sangiorgi, D.: Ensuring termination by typability, *Inf. Comput.*, Vol. 204, No. 7(2006), pp. 1045–1082.

[3] Gotsman, A., Cook, B., Parkinson, M. J., and Vafeiadis, V.: Proving that non-blocking algorithms don't block, *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Shao, Z. and Pierce, B. C.(eds.), ACM, 2009, pp. 16–28.

[4] Harris, T. L.: A Pragmatic Implementation of Non-blocking Linked-Lists, *Distributed Computing, 15th International Conference, DISC 2001, Lisbon, Portugal, October 3-5, 2001, Proceedings*, Welch, J. L.(ed.), Lecture Notes in Computer Science, Vol. 2180, Springer, 2001, pp. 300–314.

[5] Herlihy, M. and Shavit, N.: *The art of multiprocessor programming*, Morgan Kaufmann, 2008.

[6] Hoffmann, J., Marmar, M., and Shao, Z.: Quantitative Reasoning for Proving Lock-Freedom, *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, IEEE Computer Society, 2013, pp. 124–133.

[7] Jia, X., Li, W., and Vafeiadis, V.: Proving Lock-Freedom Easily and Automatically, *Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, Mumbai, India, January 15-17, 2015*, Leroy, X. and Tiu, A.(eds.), ACM, 2015, pp. 119–127.

[8] Kobayashi, N.: TyPiCal: A Type-Based Analyzer for the Pi-Calculus, `http://www-kb.is.s.u-tokyo.ac.jp/~koba/typical/`.

[9] Kobayashi, N.: A Type System for Lock-Free Processes, *Inf. Comput.*, Vol. 177, No. 2(2002), pp. 122–159.

[10] Kobayashi, N.: A New Type System for Deadlock-Free Processes, *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, Baier, C. and Hermanns, H.(eds.), Lecture Notes in Computer Science, Vol. 4137, Springer, 2006, pp. 233–247.

[11] Kobayashi, N., Saito, S., and Sumii, E.: An Implicitly-Typed Deadlock-Free Process Calculus, *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, Palamidessi, C.(ed.), Lecture Notes in Computer Science, Vol. 1877, Springer, 2000, pp. 489–503.

[12] Kobayashi, N. and Sangiorgi, D.: A hybrid type system for lock-freedom of mobile processes, *ACM Trans. Program. Lang. Syst.*, Vol. 32, No. 5(2010), pp. 16:1–16:49.

[13] Michael, M. M.: High performance dynamic lock-free hash tables and list-based sets, *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2002, Winnipeg, Manitoba, Canada, August 11-13, 2002*, Rosenberg, A. L. and Maggs, B. M.(eds.), ACM, 2002, pp. 73–82.

[14] Michael, M. M. and Scott, M. L.: Simple, Fast, and Practical Non-Blocking and Blocking Concurrent Queue Algorithms, *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA, May 23-26, 1996*, Burns, J. E. and Moses, Y.(eds.), ACM, 1996, pp. 267–275.

[15] Milner, R.: *Communicating and mobile systems - the Pi-calculus*, Cambridge University Press, 1999.

[16] Sangiorgi, D. and Walker, D.: *The Pi-Calculus - a theory of mobile processes*, Cambridge University Press, 2001.

[17] Sumii, E. and Kobayashi, N.: A Generalized Deadlock-Free Process Calculus, *Electron. Notes Theor. Comput. Sci.*, Vol. 16, No. 3(1998), pp. 225–247.

[18] Treiber, R. K.: Systems Programming: Coping With Parallelism, Technical report, IBM Almaden Research Center, 650 Harry RoadSan Jose, California, 1986.

[19] 佐久間亮: ロックフリーアルゴリズムの形式的検証,
修士論文, 情報科学研究科, 東北大学, 2018.

## A push クライアントの状態

```
1  P_0 =
2  new r in push!(1, r). r?rep
3
4  P_1 =
5  new r in r?rep
6  | (new ref in makeCell!ref. ref?quad'.
7     let writeData = third quad' in
8     let writeNext = fourth quad' in
9     new a in writeData!(value, a). a?l.
10    new while in while!reply
11    | *while?r. stack?t. ( stack!t
12      | new a in writeNext!(t, a). a?l. lock?l.
13        stack?s. ( stack!s
14        | if s = t
15          then stack?u. ( stack!quad' | lock!0 | r!0 )
16          else ( lock!0 | while!r ))))
17
18 P_2 =
19 new r in new ref in
20 r?rep
21 | (new readData in new readNext in new writeData in new writeNext in
22    let quad = makeQuadruple (readData, readNext, writeData, writeNext) in
23    r!quad | new data in new next in data!0 | next!null
24    | *readData?r. data?v. (r!v | data!v)
25    | *readNext?r. next?v. (r!v | next!v)
26    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
27    | *writeNext?(v2,r). next?v1. (r!0 | next!v2))
28 | (ref?quad'.
29    let writeData = third quad' in
30    let writeNext = fourth quad' in
31    new a in writeData!(value, a). a?l.
32    new while in while!reply
33    | *while?r. stack?t. ( stack!t
34      | new a in writeNext!(t, a). a?l. lock?l.
35        stack?s. ( stack!s
36        | if s = t
37          then stack?u. ( stack!quad' | lock!0 | r!0 )
38          else ( lock!0 | while!r ))))
39
40 P_3 =
41 new r in new ref in
42 new readData in new readNext in new writeData in new writeNext in
43 r?rep
```

```
44 | (new data in new next in data!0 | next!null
45    | *readData?r. data?v. (r!v | data!v)
46    | *readNext?r. next?v. (r!v | next!v)
47    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
48    | *writeNext?(v2,r). next?v1. (r!0 | next!v2))
49 | (let writeData = third quad' in
50    let writeNext = fourth quad' in
51    new a in writeData!(1, a). a?l.
52    new while in while!reply
53    | *while?r. stack?t. ( stack!t
54      | new a in writeNext!(t, a). a?l. lock?l.
55          stack?s. ( stack!s
56          | if s = t
57            then stack?u. ( stack!quad' | lock!0 | r!0 )
58            else ( lock!0 | while!r ))))
59
```

60 $P_4 =$

```
61 new r in new ref in new a1 in
62 new readData in new readNext in new writeData in new writeNext in
63 r?rep
64 | (new data in new next in data!0 | next!null
65   | *readData?r. data?v. (r!v | data!v)
66   | *readNext?r. next?v. (r!v | next!v)
67   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
68   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
69   | data?v1. (a1!0 | data!1))
70 | (a1?l.
71    new while in while!reply
72    | *while?r. stack?t. ( stack!t
73      | new a in writeNext!(t, a). a?l. lock?l.
74        stack?s. ( stack!s
75        | if s = t
76          then stack?u. ( stack!quad' | lock!0 | r!0 )
77          else ( lock!0 | while!r ))))
78
```

79 $P_5 =$

```
80 new r in new ref in new a1 in
81 new readData in new readNext in new writeData in new writeNext in
82 r?rep
83 | (new data in new next in next!null
84   | *readData?r. data?v. (r!v | data!v)
85   | *readNext?r. next?v. (r!v | next!v)
86   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
87   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
88   | (r!0 | data!1))
```

```
89  | (a1?l.
90     new while in while!reply
91     | *while?r. stack?t. ( stack!t
92       | new a in writeNext!(t, a). a?l. lock?l.
93         stack?s. ( stack!s
94         | if s = t
95           then stack?u. ( stack!quad' | lock!0 | r!0 )
96           else ( lock!0 | while!r ))))
97
```

98 $P_6 =$

```
99  new r in new ref in new a1 in
100 new readData in new readNext in new writeData in new writeNext in
101 r?rep
102 | (new data in new next in next!null
103   | *readData?r. data?v. (r!v | data!v)
104   | *readNext?r. next?v. (r!v | next!v)
105   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
106   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
107   | data!1)
108 | (new while in while!reply
109    | *while?r. stack?t. ( stack!t
110      | new a in writeNext!(t, a). a?l. lock?l.
111        stack?s. ( stack!s
112        | if s = t
113          then stack?u. ( stack!quad' | lock!0 | r!0 )
114          else ( lock!0 | while!r ))))
115
```

116 $P_7 =$

```
117 new r in new ref in new a1 in
118 new readData in new readNext in new writeData in new writeNext in
119 r?rep
120 | (new data in new next in next!null
121   | *readData?r. data?v. (r!v | data!v)
122   | *readNext?r. next?v. (r!v | next!v)
123   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
124   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
125   | data!1)
126 | (new while in
127    | *while?r. stack?t. ( stack!t
128       | new a in writeNext!(t, a). a?l. lock?l.
129         stack?s. ( stack!s
130         | if s = t
131           then stack?u. ( stack!quad' | lock!0 | r!0 )
132           else ( lock!0 | while!r )))
133    | stack?t. ( stack!t
```

```
134      | new a in writeNext!(t, a). a?l. lock?l.
135        stack?s. ( stack!s
136        | if s = t
137          then stack?u. ( stack!quad’ | lock!0 | r!0 )
138          else ( lock!0 | while!r ))))
139
```

140 $P_8 =$

```
141 new r in new ref in new a1 in
142 new readData in new readNext in new writeData in new writeNext in
143 r?rep
144 | (new data in new next in next!null
145   | *readData?r. data?v. (r!v | data!v)
146   | *readNext?r. next?v. (r!v | next!v)
147   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
148   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
149   | data!1)
150 | (new while in
151    | *while?r. stack?t. ( stack!t
152       | new a in writeNext!(t, a). a?l. lock?l.
153         stack?s. ( stack!s
154         | if s = t
155           then stack?u. ( stack!quad’ | lock!0 | r!0 )
156           else ( lock!0 | while!r )))
157    | ( stack!quad
158      | new a in writeNext!(quad, a). a?l. lock?l.
159        stack?s. ( stack!s
160        | if s = quad
161          then stack?u. ( stack!quad’ | lock!0 | r!0 )
162          else ( lock!0 | while!r ))))
163
```

164 $P_9 =$

```
165 new r in new ref in new a1 in new a2 in
166 new readData in new readNext in new writeData in new writeNext in
167 r?rep
168 | (new data in new next in next!null
169   | *readData?r. data?v. (r!v | data!v)
170   | *readNext?r. next?v. (r!v | next!v)
171   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
172   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
173   | data!1
174   | next?v1. (a2!0 | next!v2))
175 | (new a in new while in
176    | *while?r. stack?t. ( stack!t
177       | new a in writeNext!(t, a). a?l. lock?l.
178         stack?s. ( stack!s
```

```
179         | if s = t
180           then stack?u. ( stack!quad' | lock!0 | r!0 )
181           else ( lock!0 | while!r )))
182    | ( stack!quad
183      | a2?l. lock?l.
184        stack?s. ( stack!s
185        | if s = quad
186          then stack?u. ( stack!quad' | lock!0 | r!0 )
187          else ( lock!0 | while!r ))))
188
```

189 $P_{10} =$

```
190 new r in new ref in new a1 in new a2 in
191 new readData in new readNext in new writeData in new writeNext in
192 r?rep
193 | (new data in new next in
194   | *readData?r. data?v. (r!v | data!v)
195   | *readNext?r. next?v. (r!v | next!v)
196   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
197   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
198   | data!1
199   | (a2!0 | next!quad))
200 | (new while in
201    | *while?r. stack?t. ( stack!t
202      | new a in writeNext!(t, a). a?l. lock?l.
203        stack?s. ( stack!s
204        | if s = t
205          then stack?u. ( stack!quad' | lock!0 | r!0 )
206          else ( lock!0 | while!r )))
207    | ( stack!quad
208      | a?l. lock?l.
209        stack?s. ( stack!s
210        | if s = quad
211          then stack?u. ( stack!quad' | lock!0 | r!0 )
212          else ( lock!0 | while!r ))))
213
```

214 $P_{11} =$

```
215 new r in new ref in new a1 in new a2 in
216 new readData in new readNext in new writeData in new writeNext in
217 r?rep
218 | (new data in new next in
219   | *readData?r. data?v. (r!v | data!v)
220   | *readNext?r. next?v. (r!v | next!v)
221   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
222   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
223   | data!1
```

```
224   | next!quad)
225 | (new while in
226    | *while?r. stack?t. ( stack!t
227       | new a in writeNext!(t, a). a?l. lock?l.
228         stack?s. ( stack!s
229         | if s = t
230           then stack?u. ( stack!quad' | lock!0 | r!0 )
231           else ( lock!0 | while!r )))
232    | ( stack!quad
233       | lock?l.
234         stack?s. ( stack!s
235         | if s = quad
236           then stack?u. ( stack!quad' | lock!0 | r!0 )
237           else ( lock!0 | while!r ))))
238
```

239 $P_{12}$ =

```
240 new r in new ref in new a1 in new a2 in
241 new readData in new readNext in new writeData in new writeNext in
242 r?rep
243 | (new data in new next in
244   | *readData?r. data?v. (r!v | data!v)
245   | *readNext?r. next?v. (r!v | next!v)
246   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
247   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
248   | data!1
249   | next!quad)
250 | (new while in
251    | *while?r. stack?t. ( stack!t
252       | new a in writeNext!(t, a). a?l. lock?l.
253         stack?s. ( stack!s
254         | if s = t
255           then stack?u. ( stack!quad | lock!0 | r!0 )
256           else ( lock!0 | while!r )))
257    | (stack!quad
258       | stack?s. ( stack!s
259         | if s = quad
260           then stack?u. ( stack!quad' | lock!0 | r!0 )
261           else ( lock!0 | while!r ))))
262
```

263 $P_{13}$ =

```
264 new r in new ref in new a1 in new a2 in
265 new readData in new readNext in new writeData in new writeNext in
266 r?rep
267 | (new data in new next in
268   | *readData?r. data?v. (r!v | data!v)
```

```
269    | *readNext?r. next?v. (r!v | next!v)
270    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
271    | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
272    | data!1
273    | next!quad)
274  | (new while in
275     | *while?r. stack?t. ( stack!t
276        | new a in writeNext!(t, a). a?l. lock?l.
277          stack?s. ( stack!s
278          | if s = t
279            then stack?u. ( stack!quad | lock!0 | r!0 )
280            else ( lock!0 | while!r )))
281     | (stack!quad''
282          | if quad'' = quad
283            then stack?u. ( stack!quad' | lock!0 | r!0 )
284            else ( lock!0 | while!r )))
285
```

286 $P_{14}$ =

```
287  new r in new ref in new a1 in new a2 in
288  new readData in new readNext in new writeData in new writeNext in
289  r?rep
290  | (new data in new next in
291    | *readData?r. data?v. (r!v | data!v)
292    | *readNext?r. next?v. (r!v | next!v)
293    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
294    | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
295    | data!1
296    | next!quad)
297  | (new while in
298     | *while?r. stack?t. ( stack!t
299     | new a in writeNext!(t, a). a?l. lock?l.
300       stack?s. ( stack!s
301       | if s = t
302         then stack?u. ( stack!quad | lock!0 | r!0 )
303         else ( lock!0 | while!r )))
304     | ( stack!quad'' | stack?u. ( stack!quad' | lock!0 | r!0 ) ))
305
```

306 $P_{15}$ =

```
307  new r in new ref in new a1 in new a2 in
308  new readData in new readNext in new writeData in new writeNext in
309  r?rep
310  | (new data in new next in
311    | *readData?r. data?v. (r!v | data!v)
312    | *readNext?r. next?v. (r!v | next!v)
313    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
```

```
314   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
315   | data!1
316   | next!quad)
317 | (new while in
318    | *while?r. stack?t. ( stack!t
319       | new a in writeNext!(t, a). a?l. lock?l.
320         stack?s. ( stack!s
321         | if s = t
322           then stack?u. ( stack!quad | lock!0 | r!0 )
323           else ( lock!0 | while!r )))
324    | ( stack!quad' | lock!0 | r!0 ))
325
```
326 $P_{16} =$

```
327 new r in new ref in new a1 in new a2 in
328 new readData in new readNext in new writeData in new writeNext in
329 (new data in new next in
330   | *readData?r. data?v. (r!v | data!v)
331   | *readNext?r. next?v. (r!v | next!v)
332   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
333   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
334   | data!1
335   | next!quad)
336 | (new while in
337    | *while?r. stack?t. ( stack!t
338       | new a in writeNext!(t, a). a?l. lock?l.
339         stack?s. ( stack!s
340         | if s = t
341           then stack?u. ( stack!quad | lock!0 | r!0 )
342           else ( lock!0 | while!r )))
343    | 0
344
```
345 $P_{-1} =$

```
346 new r in new ref in new a1 in new a2 in
347 new readData in new readNext in new writeData in new writeNext in
348 r?rep
349 | (new data in new next in
350   | *readData?r. data?v. (r!v | data!v)
351   | *readNext?r. next?v. (r!v | next!v)
352   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
353   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
354   | data!1
355   | next!quad)
356 | (new while in
357    | *while?r. stack?t. ( stack!t
358       | new a in writeNext!(t, a). a?l. lock?l.
```

```
359         stack?s. ( stack!s
360         | if s = t
361           then stack?u. ( stack!quad’ | lock!0 | r!0 )
362           else ( lock!0 | while!r )))
363    | while!r ))
```

## B pop クライアントの状態

```
 1 Q_0 =
 2 new r in pop!x . r?rep
 3
 4 Q_1 =
 5 new r in r?rep
 6 | (new while in while!reply
 7    | *while?r. stack?t. ( stack!t
 8      | if s = null
 9        then r!0
10        else let readData = first t in
11             let readNext = second t in
12             new a in readData!a. a?v.
13             new b in readNext!b. b?x.
14             lock?l. stack?s. ( stack!s
15             | if s = t
16               then stack?u. ( stack!x | lock!0 | r!v )
17               else ( lock!0 | while!r ))))
18
19 Q_2 =
20 new r in r?rep
21 | (new while in
22   *while?r. stack?t. ( stack!t
23      | if s = null
24        then r!0
25        else let readData = first t in
26             let readNext = second t in
27             new a in readData!a. a?v.
28             new b in readNext!b. b?x.
29             lock?l. stack?s. ( stack!s
30             | if s = t
31               then stack?u. ( stack!x | lock!0 | r!v )
32               else ( lock!0 | while!r )))
33    | (stack?t. ( stack!t
34      | if s = null
35        then r!0
36        else let readData = first t in
37             let readNext = second t in
```

```
38             new a in readData!a. a?v.
39             new b in readNext!b. b?x.
40             lock?l. stack?s. ( stack!s
41             | if s = t
42               then stack?u. ( stack!x | lock!0 | r!v )
43               else ( lock!0 | while!r )))))
44
```

45 $Q_3 =$

```
46 new r in r?rep
47 | (new while in
48   *while?r. stack?t. ( stack!t
49      | if s = null
50        then r!0
51        else let readData = first t in
52             let readNext = second t in
53             new a in readData!a. a?v.
54             new b in readNext!b. b?x.
55             lock?l. stack?s. ( stack!s
56             | if s = t
57               then stack?u. ( stack!x | lock!0 | r!v )
58               else ( lock!0 | while!r )))
59    | ( stack!quad
60      | if quad = null
61        then r!0
62        else let readData = first quad in
63             let readNext = second quad in
64             new a in readData!a. a?v.
65             new b in readNext!b. b?x.
66             lock?l. stack?s. ( stack!s
67             | if s = quad
68               then stack?u. ( stack!x | lock!0 | r!v )
69               else ( lock!0 | while!r ))))
70
```

71 $Q_4 =$

```
72 new r in r?rep
73 | (new while in
74   *while?r. stack?t. ( stack!t
75      | if s = null
76        then r!0
77        else let readData = first t in
78             let readNext = second t in
79             new a in readData!a. a?v.
80             new b in readNext!b. b?x.
81             lock?l. stack?s. ( stack!s
82             | if s = t
```

```
83              then stack?u. ( stack!x | lock!0 | r!v )
84              else ( lock!0 | while!r )))
85    | r!0)
86
87 Q5 =
88 new r in
89 | (new while in
90   *while?r. stack?t. ( stack!t
91      | if s = null
92        then r!0
93        else let readData = first t in
94             let readNext = second t in
95             new a in readData!a. a?v.
96             new b in readNext!b. b?x.
97             lock?l. stack?s. ( stack!s
98             | if s = t
99               then stack?u. ( stack!x | lock!0 | r!v )
100              else ( lock!0 | while!r )))
101   | 0)
102
103 Q6 =
104 new r in r?rep
105 | (new while in
106   *while?r. stack?t. ( stack!t
107      | if s = null
108        then r!0
109        else let readData = first t in
110             let readNext = second t in
111             new a in readData!a. a?v.
112             new b in readNext!b. b?x.
113             lock?l. stack?s. ( stack!s
114             | if s = t
115               then stack?u. ( stack!x | lock!0 | r!v )
116               else ( lock!0 | while!r )))
117   | ( stack!quad
118     | new a in readData!a. a?v.
119       new b in readNext!b. b?x.
120       lock?l. stack?s. ( stack!s
121       | if s = quad
122         then stack?u. ( stack!x | lock!0 | r!v )
123         else ( lock!0 | while!r ))))
124
125 Q7 =
126 new r in new readData in new readNext in new writeData in new writeNext in
127 new a in r?rep
```

```
128 | (new data in new next in data!1 | next!quad''
129    | *readData?r. data?v. (r!v | data!v)
130    | *readNext?r. next?v. (r!v | next!v)
131    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
132    | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
133    | data?v. (a!v | data!v))
134 | (new while in
135   *while?r. stack?t. ( stack!t
136      | if s = null
137        then r!0
138        else let readData = first t in
139             let readNext = second t in
140             new a in readData!a. a?v.
141             new b in readNext!b. b?x.
142             lock?l. stack?s. ( stack!s
143             | if s = t
144               then stack?u. ( stack!x | lock!0 | r!v )
145               else ( lock!0 | while!r )))
146    | ( stack!quad
147      | a?v.
148        new b in readNext!b. b?x.
149        lock?l. stack?s. ( stack!s
150        | if s = quad
151          then stack?u. ( stack!x | lock!0 | r!v )
152          else ( lock!0 | while!r ))))
153
```

154 $Q_8 =$

```
155 new r in new readData in new readNext in new writeData in new writeNext in
156 new a in r?rep
157 | (new data in new next in next!quad''
158      | *readData?r. data?v. (r!v | data!v)
159      | *readNext?r. next?v. (r!v | next!v)
160      | *writeData?(v2,r). data?v1. (r!0 | data!v2)
161      | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
162      | (a!1 | data!1))
163 | (new while in
164   *while?r. stack?t. ( stack!t
165      | if s = null
166        then r!0
167        else let readData = first t in
168             let readNext = second t in
169             new a in readData!a. a?v.
170             new b in readNext!b. b?x.
171             lock?l. stack?s. ( stack!s
172             | if s = t
```

```
173              then stack?u. ( stack!x | lock!0 | r!v )
174              else ( lock!0 | while!r )))
175    | ( stack!quad
176      | a?v.
177        new b in readNext!b. b?x.
178        lock?l. stack?s. ( stack!s
179        | if s = quad
180          then stack?u. ( stack!x | lock!0 | r!v )
181          else ( lock!0 | while!r ))))
182
```

183 $Q_9 =$

```
184 new r in new readData in new readNext in new writeData in new writeNext in
185 new a in r?rep
186 | (new data in new next in next!quad''
187       | *readData?r. data?v. (r!v | data!v)
188       | *readNext?r. next?v. (r!v | next!v)
189       | *writeData?(v2,r). data?v1. (r!0 | data!v2)
190       | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
191       | data!1)
192 | (new while in
193   *while?r. stack?t. ( stack!t
194      | if s = null
195        then r!0
196        else let readData = first t in
197             let readNext = second t in
198             new a in readData!a. a?v.
199             new b in readNext!b. b?x.
200             lock?l. stack?s. ( stack!s
201             | if s = t
202               then stack?u. ( stack!x | lock!0 | r!v )
203               else ( lock!0 | while!r )))
204    | ( stack!quad
205      | new b in readNext!b. b?x.
206        lock?l. stack?s. ( stack!s
207        | if s = quad
208          then stack?u. ( stack!x | lock!0 | r!1 )
209          else ( lock!0 | while!r ))))
210
```

211 $Q_{10} =$

```
212 new r in new readData in new readNext in new writeData in new writeNext in
213 new a in new b in r?rep
214 | (new data in new next in next!quad''
215    | *readData?r. data?v. (r!v | data!v)
216    | *readNext?r. next?v. (r!v | next!v)
217    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
```

```
218    | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
219    | data!1
220    | next?v. (b!v | next!v))
221 | (new while in
222   *while?r. stack?t. ( stack!t
223      | if s = null
224        then r!0
225        else let readData = first t in
226             let readNext = second t in
227             new a in readData!a. a?v.
228             new b in readNext!b. b?x.
229             lock?l. stack?s. ( stack!s
230             | if s = t
231               then stack?u. ( stack!x | lock!0 | r!v )
232               else ( lock!0 | while!r )))
233    | ( stack!quad
234      | b?x.
235        lock?l. stack?s. ( stack!s
236        | if s = quad
237          then stack?u. ( stack!x | lock!0 | r!1 )
238          else ( lock!0 | while!r ))))
239
```

240 $Q_{11} =$

```
241 new r in new readData in new readNext in new writeData in new writeNext in
242 new a in new b in r?rep
243 | (new data in new next in
244    | *readData?r. data?v. (r!v | data!v)
245    | *readNext?r. next?v. (r!v | next!v)
246    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
247    | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
248    | data!1
249    | (b!quad'' | next!quad''))
250 | (new while in
251   *while?r. stack?t. ( stack!t
252      | if s = null
253        then r!0
254        else let readData = first t in
255             let readNext = second t in
256             new a in readData!a. a?v.
257             new b in readNext!b. b?x.
258             lock?l. stack?s. ( stack!s
259             | if s = t
260               then stack?u. ( stack!x | lock!0 | r!v )
261               else ( lock!0 | while!r )))
262    | ( stack!quad
```

```
263      | b?x.
264        lock?l. stack?s. ( stack!s
265        | if s = quad
266          then stack?u. ( stack!x | lock!0 | r!1 )
267          else ( lock!0 | while!r ))))
268
```

269 $Q_{12} =$

```
270 new r in new readData in new readNext in new writeData in new writeNext in
271 new a in new b in r?rep
272 | (new data in new next in
273    | *readData?r. data?v. (r!v | data!v)
274    | *readNext?r. next?v. (r!v | next!v)
275    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
276    | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
277    | data!1
278    | next!quad'')
279 | (new while in
280   *while?r. stack?t. ( stack!t
281      | if s = null
282        then r!0
283        else let readData = first t in
284             let readNext = second t in
285             new a in readData!a. a?v.
286             new b in readNext!b. b?x.
287             lock?l. stack?s. ( stack!s
288             | if s = t
289               then stack?u. ( quad | lock!0 | r!v )
290               else ( lock!0 | while!r )))
291    | ( stack!quad
292      | lock?l. stack?s. ( stack!s
293        | if s = quad
294          then stack?u. ( stack!quad | lock!0 | r!1 )
295          else ( lock!0 | while!r ))))
296
```

297 $Q_{13} =$

```
298 new r in new readData in new readNext in new writeData in new writeNext in
299 new a in new b in r?rep
300 | (new data in new next in
301   | *readData?r. data?v. (r!v | data!v)
302   | *readNext?r. next?v. (r!v | next!v)
303   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
304   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
305   | data!1
306   | next!quad'')
307 | (new while in
```

```
308   *while?r. stack?t. ( stack!t
309      | if s = null
310        then r!0
311        else let readData = first t in
312             let readNext = second t in
313             new a in readData!a. a?v.
314             new b in readNext!b. b?x.
315             lock?l. stack?s. ( stack!s
316             | if s = t
317               then stack?u. ( stack!x | lock!0 | r!v )
318               else ( lock!0 | while!r )))
319    | ( stack!quad
320      | stack?s. ( stack!s
321        | if s = quad
322          then stack?u. ( stack!quad | lock!0 | r!1 )
323          else ( lock!0 | while!r ))))
324
```

325 $Q_{14} =$

```
326 new r in new readData in new readNext in new writeData in new writeNext in
327 new a in new b in r?rep
328 | (new data in new next in
329   | *readData?r. data?v. (r!v | data!v)
330   | *readNext?r. next?v. (r!v | next!v)
331   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
332   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
333   | data!1
334   | next!quad'')
335 | (new while in
336   *while?r. stack?t. ( stack!t
337      | if s = null
338        then r!0
339        else let readData = first t in
340             let readNext = second t in
341             new a in readData!a. a?v.
342             new b in readNext!b. b?x.
343             lock?l. stack?s. ( stack!s
344             | if s = t
345               then stack?u. ( stack!x | lock!0 | r!v )
346               else ( lock!0 | while!r )))
347    | ( stack!quad'
348        | if quad' = quad
349          then stack?u. ( stack!quad | lock!0 | r!v )
350          else ( lock!0 | while!r )))
351
```

352 $Q_{15} =$

```
353 new r in new readData in new readNext in new writeData in new writeNext in
354 new a in new b in r?rep
355 | (new data in new next in
356   | *readData?r. data?v. (r!v | data!v)
357   | *readNext?r. next?v. (r!v | next!v)
358   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
359   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
360   | data!1
361   | next!quad'')
362 | (new while in
363   *while?r. stack?t. ( stack!t
364      | if s = null
365        then r!0
366        else let readData = first t in
367             let readNext = second t in
368             new a in readData!a. a?v.
369             new b in readNext!b. b?x.
370             lock?l. stack?s. ( stack!s
371             | if s = t
372               then stack?u. ( stack!x | lock!0 | r!v )
373               else ( lock!0 | while!r )))
374    | stack!quad'
375      | stack?u. ( stack!quad | lock!0 | r!1 )))
376
```

377 $Q_{16} =$

```
378 new r in new readData in new readNext in new writeData in new writeNext in
379 new a in new b in r?rep
380 | (new data in new next in
381   | *readData?r. data?v. (r!v | data!v)
382   | *readNext?r. next?v. (r!v | next!v)
383   | *writeData?(v2,r). data?v1. (r!0 | data!v2)
384   | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
385   | data!1
386   | next!quad'')
387 | (new while in
388   *while?r. stack?t. ( stack!t
389      | if s = null
390        then r!0
391        else let readData = first t in
392             let readNext = second t in
393             new a in readData!a. a?v.
394             new b in readNext!b. b?x.
395             lock?l. stack?s. ( stack!s
396             | if s = t
397               then stack?u. ( stack!x | lock!0 | r!v )
```

```
398              else ( lock!0 | while!r )))
399    | ( stack!quad | lock!0 | r!1 ))
400
```

401 $Q_{17}$ =

```
402 new r in new readData in new readNext in new writeData in new writeNext in
403 new a in new b in r?rep
404 | (new data in new next in
405    | *readData?r. data?v. (r!v | data!v)
406    | *readNext?r. next?v. (r!v | next!v)
407    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
408    | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
409    | data!1
410    | next!quad'')
411 | (new while in
412   *while?r. stack?t. ( stack!t
413      | if s = null
414        then r!0
415        else let readData = first t in
416             let readNext = second t in
417             new a in readData!a. a?v.
418             new b in readNext!b. b?x.
419             lock?l. stack?s. ( stack!s
420             | if s = t
421               then stack?u. ( stack!x | lock!0 | r!v )
422               else ( lock!0 | while!r )))
423    | 0)
424
```

425 $Q_{-1}$ =

```
426 new r in new readData in new readNext in new writeData in new writeNext in
427 new a in new b in r?rep
428 | (new data in new next in
429    | *readData?r. data?v. (r!v | data!v)
430    | *readNext?r. next?v. (r!v | next!v)
431    | *writeData?(v2,r). data?v1. (r!0 | data!v2)
432    | *writeNext?(v2,r). next?v1. (r!0 | next!v2)
433    | data!1
434    | next!quad'')
435 | (new while in
436   *while?r. stack?t. ( stack!t
437      | if s = null
438        then r!0
439        else let readData = first t in
440             let readNext = second t in
441             new a in readData!a. a?v.
442             new b in readNext!b. b?x.
```

```
443            lock?l. stack?s. ( stack!s
444            | if s = t
445              then stack?u. ( stack!x | lock!0 | r!v )
446              else ( lock!0 | while!r )))
447    | while!r ))
```