プログラミング－言語・基礎・実践－　5－4
(1991. 11. 21)

接続法による並列処理言語DNNPの表示的意味記述について

大山口　通夫　　山田　高明　　太田　義勝

三重大学　工学部

チャネルを用いた非同期通信、非決定的選択、並列実行、動的なプロセス生成などの基本的機能を有する並列処理言語に対して、了解性の高い表示的(denotational)意味記述が那須、大山口(1989)によって与えられた。しかしながら、その意味記述においては、入力待ちの発生場所の記憶、その時点の環境の退避、さらにその発生場所の探索といった操作的な面を含んでいるという問題点があった。

そこで本研究では、この問題点を解決するためにgoto文を持つプログラム言語の意味記述に用いられてきた接続法 (continuation) の概念に着目し、この概念を用いた並列処理言語の意味記述法について考察した。その結果、上記の並列処理言語に対してより抽象度の高い、見通しの良い表示的意味記述を得ることができたのでその概要について報告する。

Continuation Semantics of Dynamic Networks of Nondeterministic Processes

Michio Oyamaguchi　Takaaki Yamada　Yoshikatsu Ohta

Faculty of Engineering, Mie University

Nasu and Oyamaguchi (1989) gave the denotational semantics of a parallel processing language, called Dynamic Networks of Nondeterministic Processes (DNNP), which can be used to describe a network of nondeterministic processes communicating asynchronously via channels. However, part of the semantics was defined in some operational way, e.g., operations for storing and looking up waiting points for input data sent via channels were explicitly used, so that the semantics did not succeed in attaining the higt degree of abstraction.

In this paper we consider to imrove the semantics given by Nasu and Oyamaguchi.Using the notaion of continuation used in the denotational semantics of sequential languages with goto satements, we present a more abstract and comprehensible continuation semantics of the language DNNP.

1.はじめに

近年、様々な並列処理言語が提案されるとともに、その形式的意味記述に関する研究が盛んに行なわれてきた。Kahn[9]はチャネルを用いたプロセス間の非同期通信に基づく並列処理言語 dDN(deterministic Dataflow Networks)を提案し、その意味記述を与えた。この言語は非決定的選択の機能を持たないために、その機能を付け加えた場合の意味記述を与えることが望まれていた。しかしながら、言語が非決定的選択の機能を持つとき、決定性の場合の意味記述の単純な拡張として意味を与えることは Brock、Ackerman[11]によって指摘されたように非常に困難である。この Brock らの問題点を克服した表示的意味記述は Kok[13]、Jonsson[14]によって与えられた。なお、これらの並列処理言語は動的プロセス生成の機能を持たない。

また、並列処理言語 dDN に動的プロセス生成の機能を加えた言語 DNP(Dynamic Networks of Processes)が Kahn ら[10]によって提案され、de Bruin[12]によってその言語が形式化され意味記述が与えられた。ただし、この言語 DNP は非決定的選択の機能を持たない言語である。

一方、言語 dDN に非決定的選択と動的プロセス生成の機能を追加した言語 DNNP に対して表示的意味を与えることは、かなり難しく幾つかの研究が行なわれてきたが不十分である。例えば[2, 3]では了解性の高い表示的(denotational)意味記述を与えてはいるものの、チャネル通信において入力待ちが発生した場合、その意味記述では入力待ちの発生場所の記憶、その時点の環境の退避、さらにその発生場所の探索といった操作的な面を含んでいるという問題点があった。

本研究では、この問題点を解決するために、goto 文を持つプログラム言語の表示的意味記述に用いられてきた接続法 (continuation)[1,4] の概念に着目し、この概念を用いてより抽象度の高い、見通しの良い表示的意味記述を与える。以下、その概要について報告する

2.並列処理言語

本稿で考察する並列処理言語は文献[2,3]で導入されたものであり、de Bruin[12]によって形式化された言語に非決定的選択の機能を付け加えたものとほぼ同じものである。以下、この言語を DNNP と呼ぶ。図 1 にこの言語の構文を示す。

```
⟨ プログラム ⟩ ::= program ⟨ id ⟩ ⟨ parm ⟩ ; ⟨ プロセス宣言 ⟩ ; ··· ; ⟨ ブロック ⟩
⟨ プロセス宣言 ⟩ ::= process ⟨ id ⟩ ⟨ parm ⟩ ; ⟨ ブロック ⟩
⟨ ブロック ⟩ ::= var ⟨ parmlist ⟩ ; ⟨ 文 ⟩
⟨parm⟩ ::= () | ( ⟨ parmlist ⟩ )
⟨parmlist⟩ ::= ⟨ id ⟩ , ··· : ⟨ 型 ⟩ | ⟨ id ⟩ , ··· : ⟨ 型 ⟩ ; ⟨ parmlist ⟩
⟨ 文 ⟩ ::= ⟨ 文1 ⟩ ; ⟨ 文2 ⟩ | if ⟨ 式 ⟩ then ⟨ 文1 ⟩ else ⟨ 文2 ⟩ | while ⟨ 式 ⟩ do ⟨ 文 ⟩
      | (⟨ var-id ⟩ , ··· , ⟨ var-id ⟩) := (⟨ 式 ⟩ ,···, ⟨ 式 ⟩)        /* 並列代入 */
      | ⟨ ラベル ⟩ : ⟨ inpch-id ⟩ ? ⟨ var-id ⟩                          /* チャネルからの入力 */
      | ⟨ outch-id ⟩ ! ⟨ 式 ⟩                                          /* チャネルへの出力 */
      | chan ⟨ inpch-id ⟩ ← ⟨ outch-id ⟩ : ⟨ 文 ⟩                      /* チャネル結合 */
      | ⟨ ラベル ⟩ : ⟨process-id⟩ ⟨parm⟩                               /* プロセス呼び出し */
      | ⟨ 文1 ⟩ // ⟨ 文2 ⟩                                            /* 並列実行 */
      | ⟨ 文1 ⟩ [] ⟨ 文2 ⟩                                            /* 非決定的選択 */
```

図 1 . 構文の概略

ここで< var-id >はブロックで宣言されたチャネル型以外の変数名を示す。< inpch-id >と< outch-id >はそれぞれ入力チャネル変数名と出力チャネル変数名であり、< process-id >はプロセス変数名である。

本稿におけるプロセス間の通信方法は非同期のメッセージ通信である。入力·出力チャネル変数を結合し、このチャネル変数をプロセスの引数とすることにより、プロセス間の通信を実現している。

プロセスはプロセス呼び出し文により動的に生成、実行される。　なお、変数に関する制限事項として、

- 並列実行文で ⟨ 文1 ⟩ と ⟨ 文2 ⟩ の両方で同じ変数を使用してはならない
- チャネル結合文で用いるチャネル変数はそのプロセスで宣言された局所変数に限る

などがある。前者は通信がチャネルを通してのみ行なわれ、共有メモリを許していないからである。後者はそのプロセスを生成した親で結合されていたチャネル変数を、再び結合することを避けている。また、各チャネル入力文とプロセス呼び出し文にはそれぞれ異なるラベル名が割り当てられているとする。詳細については後述する。図 2

にこの並列処理言語を用いた簡単な例を示す。

```
program  Example() ;

  process  P(c : inpch) ;
   var   x , ··· : integer ;                    プロセスP の宣言
         ··· ; l : c ? x ; ···

  process  Q(d : outch) ;
   var   y , ··· : integer ;                    プロセスQ の宣言
         ··· ; d ! y ; ···

  var   c : inpch;  d : outch;  p : P;  q : Q;
        ···
         chan   c ← d :                         プログラムExample のメイン = 初期プロセス
                ··· ;  p(c)  //  q(d)  ; ···
        ···
```

図2．チャネルを通したプロセス間通信の例

3.表示的意味記述

本稿の意味記述では次の記法を用いる。

〈記法〉　X, Y：集合、状態$\sigma \in S$、ローカル環境$\rho \in Lenv$、ロケーション$l \in Location$、$x, y \in Id$としたとき、

- $X \to Y$はXからYへのすべての関数の集合を表す。
  また、完備束X、Yに対し、$[\ X \to Y\ ]$はXからYへの連続関数全体の集合を表現する。
- $\to$演算子は右結合である。(例: $A \to B \to C \to D = A \to (B \to (C \to D))$)
- 関数は左結合で適用される。(例: $fabc = ((f(a))(b))(c)$)
- $\sigma.\rho.x\ = \sigma(\rho(x))$
- $\rho[l/x].y\ = \begin{cases} l & \text{if} \quad x = y \\ \rho.y & \text{otherwise} \end{cases}$
- $\rho[< l_1, \cdots, l_n > / < x_1, \cdots, x_n >]\ = \rho[\ l_1/x_1,\ \cdots\ , l_n/x_n\ ]$

3.1.意味領域

プログラムの意味とはプログラムを実行させたときに生じる効果であるといえる。すなわち、変数の値の変化と考えられ、状態を変数名からその値への写像と定義したとき、プログラムの意味は状態から状態への関数とみなすことができる。

状態を変数名から値への写像であると定義したが、図1における言語で用いられる変数は動的な存在であり、プロセス内のローカル変数は呼び出しのたびに記憶場所を割り当てられる。ゆえに、同じ変数が各レベルで別々に生成される。そのため、変数Idからその記憶場所$Location$を与える環境と、$Location$からそのロケーション（記憶場所）に格納されている値$Value$を返す状態Sを導入して、二段階に分ける。ここで、環境は変数名からそのロケーションを与えるローカル環境$Lenv$と、プロセス名からそのプロセス関数である$Proc_func$を与えると共にラベルと入力チャネル変数の対から対応する接続である$Prog_func$を与えるグローバル環境$Genv$に分ける。

命令接続Cは状態を受けとり、状態の集合$P(S)$を与える。これは非決定的選択文が複数の状態を返すことを考慮したものであり、この集合$P(S)$は[6,7]のPower-domainと同様な集合とする。これにより領域及びその上での連続関数の存在が保証できる。また、式接続Kは値$Value$の直積を受けとり、命令接続を返す関数の集合である。

文の意味関数は文と環境、さらに、受けとった文以降の効果を示す命令接続を受けとったときに、その文からの効果を表す命令接続を与える関数である。プロセスの意味関数は引数のロケーションとローカル環境、命令接続を与えたときに、命令接続を返す関数である。プロセス宣言・ラベルの意味関数はプログラムが与えられたときに、プロセス名とその意味を表す$Proc_func$の対と、ラベルとチャネル変数の対から対応する意味を求める$Prog_func$の対をグローバル環境に埋め込み、グローバル環境の更新を行なう。

これらの意味領域を図3に示す。

文の意味関数	$\mathcal{C}$	$\in$	[文$\rightarrow Genv \rightarrow Lenv \rightarrow C \rightarrow C$]
式の意味関数	$\mathcal{E}$	$\in$	[式$\times\cdots\times$式$\rightarrow Lenv \rightarrow K \rightarrow C$]
グローバル環境	g	$\in$	$Genv = [Id \rightarrow Proc\_func] \cup [Id \times Id \rightarrow Prog\_func]$
	$Proc\_func$	$=$	$\cup_{m\geq 0}[Location^m \rightarrow Lenv \rightarrow C \rightarrow C]$
	$Prog\_func$	$=$	$[Lenv \rightarrow C \rightarrow C]$
ローカル環境	ρ	$\in$	$Lenv = [Id \rightarrow Location]$
状態	σ	$\in$	$S = [Location \rightarrow Value]$
命令接続	θ	$\in$	$C = [S \rightarrow \mathcal{P}(S)]$
式接続	κ	$\in$	$K = [Value \times \cdots \times Value \rightarrow C]$
宣言接続	χ	$\in$	$D = [Lenv \rightarrow C]$
プログラム	$\mathcal{P}rg$	$\in$	[プログラム$\rightarrow Genv \rightarrow Lenv \rightarrow C \rightarrow C$]
プロセス宣言･ラベル	$\mathcal{P}l$	$\in$	[プログラム$\rightarrow Genv \rightarrow Genv$]
ブロック	$\mathcal{B}lk$	$\in$	[ブロック$\rightarrow Genv \rightarrow Lenv \rightarrow C \rightarrow C$]
変数宣言	$\mathcal{D}ec$	$\in$	[変数宣言$\rightarrow Lenv \rightarrow D \rightarrow C$]
ラベル接続	$\mathcal{L}, \mathcal{L}_p, \mathcal{L}_c$	$\in$	$[Id \times Id \rightarrow Prog\_frag \rightarrow Genv \rightarrow Lenv \rightarrow C \rightarrow C]$

図３．意味領域

次に、意味領域での半順序を定義する。$\perp$ (bottom)を含まない値の集合をDとし、Dに$\perp$を加えた集合を$D^{\perp}$と定義する。この$\perp$の意味は未定義であり、プログラムの意味が$\perp$であるとは、プログラムの実行が結果を出さずに止まらないことに対応する。要素$d_1, d_2 \in D^{\perp}$に対して、半順序$\sqsubseteq$は以下によって定義される。

$$d_1 \sqsubseteq d_2 \Leftrightarrow (d_1 = d_2) \vee (d_1 = \perp)$$

チャネルで用いられるデータはストリームと呼ばれるメッセージの有限、あるいは無限の列によって構成される。Dの要素によって、有限の列を構成したものを要素とする集合に対してD^*、無限の列を要素とする集合をD^{∞}とする。ストリームの集合は以下のように定義される。

$$STREAM(D) = (D^* \cdot \{\perp\}) \cup D^* \cup D^{\infty}$$

ここで $\cdot$ はストリームの連結を表す。ストリーム$s_1, s_2 \in STREAM(D)$に対する半順序は次のように定義する。

$$s_1 \sqsubseteq s_2 \Leftrightarrow s_1 = s_2 \vee (\exists s_3 \in D^*,\ s_4 \in STREAM(D) : (s_1 = s_3 \cdot \perp) \wedge (s_2 = s_3 \cdot s_4))$$

ここで$(D^{\perp}, \sqsubseteq)$ 及び $(STREAM(D), \sqsubseteq)$は明らかに完全半順序である。また、$Location$などに対する半順序も$D^{\perp}$と同様に定義される。

以上の要素間の半順序は関数間の半順序に自然に拡張される。

$f, g \in [D_1 \rightarrow D_2]$に対して、　　$f \sqsubseteq g \Leftrightarrow (\forall d \in D_1 : f(d) \sqsubseteq g(d))$

最後に、今回用いる補助関数を定義する。ストリーム$s, s_1, s_2 \in STREAM(D)$に対して、

$$\mathtt{append}(\epsilon, s) = s$$

$$\mathtt{append}((a \cdot s_1), s_2) = \begin{cases} a \cdot (\mathtt{append}(s_1, s_2)) & \text{if} \quad a \neq \perp \\ \perp & \text{if} \quad a = \perp \end{cases}$$

3.2.意味関数

接続法とはある指定されたプログラム部分以降のプログラムの実行の効果を表現する手段である。一般的に接続は状態から状態への関数として定義され、接続法における文の意味とは、ある文とその文以降の効果を表す接続θからその文をも含めた効果である接続θ'を返す関数として与えられる。これを形式的に記述すると、

$$\mathcal{C}[\![\ 文\ ]\!]\rho\theta = \theta'$$

となる。接続θとθ'の関係を状態を用いて説明すると、ある文を実行する前の状態がσ_0、実行後の状態をσ_1であり、それ以降の接続θにより状態がσ_2に変化するならば、接続θ'とは状態σ_0を受けとりσ_2を返す関数である。なお、本稿では接続を状態から状態の集合への関数として定義した。これは前述したように、非決定的選択文における複数の結果を考慮したためである。

プロセス間通信で入力待ちが発生したときを図２の例を図４に図示し、直感的に説明する。

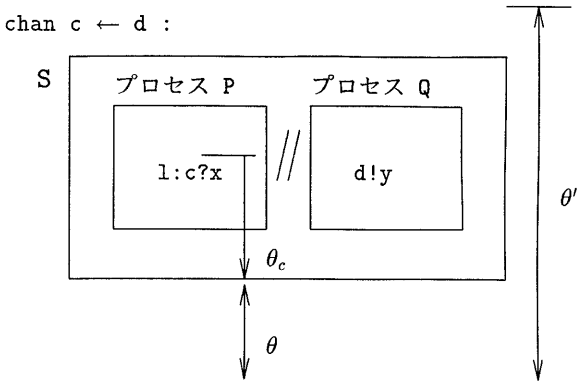

図4．チャネルを通したプロセス間通信

プロセスPのチャネル入力文c?xでプロセスQからのストリームが得られないときには、プロセスPの実行はc?xで待たされ、ストリームが得られたときにはc?xから再開される。今回、与えた意味記述は入力待ちが発生した際には、c?xからチャネル結合文の最後までの接続θ_cをcontなるフィールドに格納してチャネル結合文の最後までスキップする。ここで、出力チャネル変数dに対する出力が得られているときには、c?xに対する入力待ちを解消し、残りの文の実行の意味をcontに格納した接続θ_cで与える。また、文Sの最後では、再び入力チャネル変数cに対して入力待ちが発生したかどうかをチェックする。ここで、発生している場合には文Sを繰り返し実行し、そうでない場合にはチャネル結合文の実行を終える。これを形式的に記述すると、

$$
\begin{aligned}
&\mathcal{C}[\![\texttt{chan c} \leftarrow \texttt{d : S}]\!]\rho\theta = \mathcal{C}[\![\texttt{S}]\!]\rho\theta'' \quad (=\theta')\\
&\theta'' = fix(\lambda\theta''.\ \mathcal{E}[\![\, c.wait \wedge \neg \texttt{empty}(d)\,]\!]\rho\texttt{cond}(\theta_1,\theta))\\
&\theta_1 = \theta'' \circ \theta_c
\end{aligned}
$$

となる。チャネル結合文の意味$\mathcal{C}[\![\texttt{chan c} \leftarrow \texttt{d : S}]\!]\rho\theta$は、まず、Sを実行して$\theta''$に接続する。$\theta''$は入力チャネルcで待ちがあり、かつdにストリームがあるときにはθ_1に、そうでないときにはθに接続してチャネル結合文の実行を終える。接続θ_1は入力待ちが発生した際にスキップしたθ_cを接続した後、再び入力チャネル変数cに対して入力待ちが発生したかどうかをチェックするために、θ''に接続する。

また、入力チャネル変数がプロセスのパラメータとして渡され、そのパラメータを持つプロセスのチャネル入力文で待ちが発生する場合、もとのプロセス呼び出し文でcontなるフィールドに格納する接続は以下のように構成する。呼ばれたプロセス中の（待ちが発生している）チャネル入力文以降の接続と、そのプロセス呼び出し文以降の接続を合成してcontに記憶し、待ちが解消されたときには先と同様にこの合成された接続で、入力待ち解消時の意味を与える。

本稿では通信の入力待ちが発生した場合、そのチャネル入力文あるいはプロセス呼び出し文以降の接続を格納する。しかし、同じ入力チャネル変数あるいはプロセス変数は複数箇所で用いられることを許しているために、これらに一意のラベルを割り当てることによって、それぞれの格納すべき接続を識別している。

また、本稿の意味記述ではいくつかの特別な変数$wait$、$wlabel$、$lwset$、$olwset$とロケーションl_0を用意している。変数$wait$はそれまでの実行でチャネル通信の入力待ちが発生したかどうかの真偽を表し、待ちが発生していれば文をスキップするために用いる。また、$wlabel$、$lwset$、$olwset$は待ちが発生したときの接続を構成するために用いるが、詳細については後述する。ロケーションl_0は未使用のロケーションの集合を表し、プロセスが生成されたときこの集合からロケーションが取り出され、新たなローカル変数に割り当てられる。l_0から新たなローカル変数のロケーションが取り出される。

なお、入力チャネル変数は$c.wait$、$c.buff$、$c.cont$の3つのフィールドを持つ。状態とローカル環境がσ、ρであるとき、各フィールドの内容は以下のように表す。

$\sigma.\rho.c.wait \in \{\ \perp\ ,\ true\ ,\ false\ \}$	：チャネルcの入力文（またはcを引数として持つプロセス呼び出し文）で入力待ち発生の真偽値
$\sigma.\rho.c.buff \in STREAM(D)$	：チャネルの内容
$\sigma.\rho.c.cont \in C$	：チャネルcの入力文（またはcを引数として持つプロセス呼び出し文）

以降の接続

3.2.1.プログラムの意味関数$\mathcal{P}rg$　　(Δ_i : プロセス宣言, Θ : ブロック)

プログラムはプロセス宣言の列と初期プロセスを示すブロックから構成される。プログラムの意味関数$\mathcal{P}rg$はプロセス宣言・ラベルの意味関数により、プロセス宣言の列からグローバル環境に情報を埋め込み、そのグローバル環境のもとで初期ブロックをブロックの意味関数に適用したもので与えられる。

$$\mathcal{P}rg[\![\mathtt{program}\Delta_1;\cdots;\Delta_m;\Theta]\!]_g = \mathcal{B}lk[\![\Theta]\!](\mathcal{P}l[\![\mathtt{program}\Delta_1;\cdots;\Delta_m;\Theta]\!]_g)$$

3.2.2.プロセス宣言・ラベルの意味関数$\mathcal{P}l$　　(l_i : ラベル, c_i : 入力チャネル変数)

プロセス宣言・ラベルの意味関数$\mathcal{P}l$はプログラム中のプロセス宣言の列を用い、各プロセス名と、ラベルと入力チャネル変数名の対をそれぞれブロックの意味と、接続に対応づけることをグローバル環境を変更することにより行なっている。ここで、プロセスが互いに呼び合ったり、プロセス中でチャネル入力文が現れるのでその意味は最小不動点で与えられる。

$$\mathcal{P}l[\![\Psi]\!]_g \;=\; g[\gamma_1/\mathtt{P}_1,\ldots,\gamma_m/\mathtt{P}_\mathtt{m},\xi_1/(l_1,c_1),\ldots,\xi_n/(l_n,c_n)]$$

$$\begin{aligned}(\gamma_1,\ldots,\gamma_m,\xi_1,\ldots,\xi_n) = fix(\ \lambda(\gamma_1,\ldots,\gamma_m,\xi_1,\ldots,\xi_n).\\ (\lambda\mu_1\rho_1.\mathcal{B}lk[\![\Theta_1]\!]_{g'}(\rho_1[\mu_1/a_1]))\ \ ,\cdots,\ \ (\lambda\mu_m\rho_m.\mathcal{B}lk[\![\Theta_m]\!]_{g'}(\rho_m[\mu_m/a_m])),\\ \mathcal{L}(l_1,c_1)[\![\Psi]\!]_{g'}\ \ ,\cdots,\ \ \mathcal{L}(l_n,c_n)[\![\Psi]\!]_{g'})\end{aligned}$$

where

$$\begin{aligned}\Psi \;&=\; \mathtt{program}\Delta_1;\cdots;\Delta_m;\Theta\\ \Delta_i &= \mathtt{process}\ \mathtt{P}_\mathtt{i}(a_i);\Theta_i \quad 1\le i\le m\\ g' \;&=\; g[\gamma_1/\mathtt{P}_1,\ldots,\gamma_m/\mathtt{P}_\mathtt{m},\xi_1/(l_1,c_1),\ldots,\xi_n/(l_n,c_n)]\end{aligned}$$

3.2.3.ブロックの意味関数$\mathcal{B}lk$　　(δ : パラメータ , S : 文)

ブロックの意味関数$\mathcal{B}lk$は、ブロックが変数宣言と文から構成されるので、変数宣言の意味関数によりローカル環境を更新したのち、その環境のもとで文に対して文の意味関数を適用したものがブロックの意味となる。

$$\mathcal{B}lk[\![\mathtt{var}\ \delta;\mathtt{S}]\!]_g\rho\theta = \mathcal{D}ec[\![\mathtt{var}\ \delta]\!]\rho(\lambda\rho'.\mathcal{C}[\![\ \mathtt{S}\ ]\!]_g\rho'\theta\ )$$

3.2.4.変数宣言の意味関数$\mathcal{D}ec$

前述したように変数は動的な存在であり、プロセス内のローカル変数を呼び出すたびにローケーションを割り当てる必要がある。それゆえ、変数宣言の意味関数$\mathcal{D}ec$はローカル環境を更新する。ただし、変数宣言の意味関数$\mathcal{D}ec$は変数の型により定義が異なる。入力チャネル変数の場合、上で述べたように$wait$、$buff$、$cont$なるフィールドが確保され、それぞれ$false$、ϵ（空）、I（恒等写像 $\in C$）に初期化される。また、出力チャネル変数は最初は空であるようにϵ（空）、プロセス変数はそのプロセス名を値として初期化される。なお、τは変数の型を表し、inpch、outchはそれぞれ入力チャネル変数と出力チャネル変数の場合である。

$$\mathcal{D}ec[\![\mathtt{var}\ \mathtt{x}:\tau;\delta]\!]\rho\chi = \mathcal{D}ec[\![\mathtt{var}\ \mathtt{x}:\tau]\!]\rho\ \{\ \lambda\rho'.\ \mathcal{D}ec[\![\mathtt{var}\ \delta]\!]\rho'\chi\ \}$$

$$\mathcal{D}ec[\![\mathtt{var}\ \mathtt{x}_1,\mathtt{x}_2,\ldots,\mathtt{x}_n{:}\tau]\!]\rho\chi = \mathcal{D}ec[\![\mathtt{var}\ \mathtt{x}_1{:}\tau]\!]\rho\ \{\ \lambda\rho'.\ \mathcal{D}ec[\![\mathtt{var}\ \mathtt{x}_2,\ldots,\mathtt{x}_n{:}\tau]\!]\rho'\chi\ \}$$

$$\mathcal{D}ec[\![\mathtt{var}\ \mathtt{x}:\tau]\!]\rho\chi\sigma = (\lambda l_x.\ \chi\ (\rho[l_x/\mathtt{x}])\ (\sigma[removeloc(\sigma.l_0,\ l_x)/l_0,\bot/l_x])\ )(getloc(\sigma.l_0))$$

（τ: プロセス名, inpch, outch 以外の型）

$$\mathcal{D}ec[\![\mathtt{var}\ \mathtt{c}:\mathtt{inpch}]\!]\rho\chi\sigma = (\lambda\bar{l}_c.\chi(\rho[\bar{l}_c/\bar{c}])(\sigma[removeloc(\sigma.l_0,\bar{l}_c)/l_0,\langle false,\epsilon,I\rangle/\bar{l}_c]))(getloc^3(\sigma.l_0))$$

where　　$\bar{c} = \langle c.wait, c.buff, c.cont\rangle$, $\bar{l}_c = \langle l_w, l_b, l_n\rangle$

$$\mathcal{D}ec[\![\mathtt{var}\ \mathtt{x}:\mathtt{outch}]\!]\rho\chi\sigma = (\lambda l_d.\ \chi\ (\rho[l_d/\mathtt{d}])\ (\sigma[removeloc(\sigma.l_0,\ l_d)/l_0,\epsilon/l_d])\ )(getloc(\sigma.l_0))$$

$$\mathcal{D}ec[\![\mathtt{var}\ \mathtt{p}:\tau_p]\!]\rho\chi\sigma = (\lambda l_p.\ \chi\ (\rho[l_p/\mathtt{p}])\ (\sigma[removeloc(\sigma.l_0,\ l_p)/l_0,\tau_\mathtt{p}/l_p])\ )(getloc(\sigma.l_0))\quad (\ \tau_p\text{: プロセス名})$$

3.2.5.式の意味関数$\mathcal{E}$　　(E_i : 式 , $x\in Id$)

式の意味関数$\mathcal{E}$は式の直積と環境、状態を受けとったとき、次のように解釈される。式の直積の各成分E_iを値の意味関数$\mathcal{V}$によってその環境と状態のもとで評価し、得られた値の直積と状態を引数として式接続に渡す。式接続とは与えられた式以降のプログラムの実行を表す。これは式を評価して得られる値が残りのプログラムの実行で使われることを表している。

$$\mathcal{E}[\![\ (E_1,\cdots,E_n)\ ]\!]\rho\kappa\sigma = \kappa\ (\langle \mathcal{V}[\![E_1]\!]\rho\sigma,\cdots,\mathcal{V}[\![E_n]\!]\rho\sigma\rangle)(\sigma)$$
$$where$$
$$\mathcal{V}\in[\text{式}\rightarrow Lenv\rightarrow S\rightarrow Value]\ ,\ \mathcal{V}[\![x]\!]\rho\sigma=\sigma.\rho.x\ ,\ \mathcal{V}[\![\ E_1\ \mathtt{op}\ E_2\ ]\!]\rho\sigma=(\mathcal{V}[\![\ E_1\ ]\!]\rho\sigma)\ \mathtt{op}\ (\mathcal{V}[\![\ E_2\ ]\!]\rho\sigma)$$

3.2.6.文の意味関数$\mathcal{C}$

1）複文

複文は文$\mathtt{S}_1$を実行してから得られる状態において入力待ちが発生したときには、文$\mathtt{S}_2$をスキップしθに接続し、入力待ちが発生していないときは文$\mathtt{S}_2$を実行するので、命令接続$\mathcal{C}[\![\mathtt{S}_2]\!]_g\rho\theta$に接続する。

$$\mathcal{C}[\![\ \mathtt{S}_1\ ;\ \mathtt{S}_2\ ]\!]_g\rho\,\theta=\mathcal{C}[\![\mathtt{S}_1]\!]_g\rho(\ \mathcal{E}[\![wait]\!]\rho\ \mathtt{cond}\ (\theta\ ,\ \mathcal{C}[\![\mathtt{S}_2]\!]_g\rho\,\theta\ )\ )$$

2）条件文

条件式Eが真であるときには、文$\mathtt{S}_1$を実行してから条件文以降を実行するので、条件文の意味はこの場合、文$\mathtt{S}_1$と条件文以降の命令接続θから得られる命令接続$\mathcal{C}[\![\mathtt{S}_1]\!]_g\rho\theta$で与えられる。また、条件式$E$が偽であるときには、文$\mathtt{S}_2$と条件文以降の命令接続θで与えられる命令接続$\mathcal{C}[\![\mathtt{S}_2]\!]_g\rho\theta$で意味が記述されている。

$$\mathcal{C}[\![\mathtt{if}\ E\ \mathtt{then}\ \mathtt{S}_1\ \mathtt{else}\ \mathtt{S}_2]\!]_g\rho\,\theta=\mathcal{E}[\![E]\!]\rho\ \mathtt{cond}\ (\ \mathcal{C}[\![\mathtt{S}_1]\!]_g\rho\,\theta\ ,\ \mathcal{C}[\![\mathtt{S}_2]\!]_g\rho\,\theta\ )$$

3）並列代入文

並列代入文はその左辺に示された変数名の直積のロケーション$\langle\rho.\mathtt{x}_1,\cdots,\rho.\mathtt{x}_n\rangle$の各成分の値が、右辺の式を評価した値$\mathcal{E}[\![(E_1,\cdots,E_n)]\!]\rho\kappa\sigma$の各成分と対応するようσを更新する。

$$\mathcal{C}[\![(\mathtt{x}_1,\cdots,\mathtt{x}_n):=(E_1,\cdots,E_n)]\!]_g\rho\,\theta=\mathcal{E}[\![(E_1,\cdots,E_n)]\!]_\rho\kappa$$
$$where\qquad \kappa=(\ \lambda\langle\nu_1,\cdots,\nu_n\rangle\sigma.\ \theta(\sigma[\ \langle\nu_1,\cdots,\nu_n\rangle/\langle\rho.\mathtt{x}_1,\cdots,\rho.\mathtt{x}_n\rangle\ ])\ )$$

4）チャネル入力文

入力チャネルにストリームがあれば、変数xにストリームの一番目の値を代入し、入力チャネルの内容をストリームの二番目以降の値とする。もし、入力チャネルにデータがなければ、入力待ち状態になるので$wait$に$true$、また入力チャネル変数cで待ちが発生したことを示すために$c.wait$に$true$、そして$wlabel$に$wlabel\cup\{l\}$をそれぞれ代入する。$wlabel$は待ち以降の接続を構成する際に使用される。また、$c.cont$にチャネル入力文以降の接続を格納する。$lwset$は待ち状態でない状態から待ちが発生した状態に状態が変化したときにその文以降の接続を記憶するために用いる。これについてはチャネル結合文において詳しく述べる。

$$\mathcal{C}[\![l:\ \mathtt{c}\ ?\ \mathtt{x}]\!]_g\rho\,\theta=\mathcal{E}[\![\neg\mathtt{empty(c)}]\!]_\rho\ \mathtt{cond}(\ \theta'\ ,\ \theta''\ )$$
$$where$$
$$\theta'=\ \mathcal{C}[\![\ (\mathtt{x},c.buff):=(\mathtt{first}(c.buff),\mathtt{rest}(c.buff))\ ]\!]_g\rho\ \theta$$
$$\theta''=\lambda\sigma.\ \mathcal{C}[\![\ (wait,c.wait,wlabel):=$$
$$(true,true,wlabel\cup\{l\})\ ]\!]_g\rho\,\theta(\sigma[g(l,\mathtt{c})\rho/\rho.c.cont,\sigma.\rho.lwset\cup\{\rho.c.cont\}/\rho.lwset])$$

$$\mathcal{E}[\![\neg\mathtt{empty(c)}]\!]_\rho\kappa\sigma=\kappa(\sigma.\rho.c.buff\ \neq\ \epsilon)\ (\sigma)$$

5）チャネル出力文

チャネル出力文は、現在の出力チャネルの内容に式Eの値を付け加えた列としてdを更新するように意味が記述されている。

$$\mathcal{C}[\![\mathtt{d}\ !\ E]\!]_g\rho\ \theta=\mathcal{E}[\![E]\!]_\rho(\ \lambda\nu\sigma.\ \theta\sigma[\ \mathtt{append}(\sigma.\rho.\mathtt{d},\ \nu)/\rho.\mathtt{d}\ ]\ )\ )$$

6）チャネル結合文

チャネル結合文は、まず$olwset$、$lwset$をϕ（空集合）に初期化したのち、文Sを実行してθ'に接続する。θ'は入力チャネル変数cで待ちが発生しており、かつ出力チャネル変数dにストリームがあるとき接続θ_1、そうでないとき接続θ_2とみなせる。θ_1は、cで待ちが発生しているのでそれ以降の文をスキップしてSの実行が終了した時点でdにデータがある場合を考慮している。すなわち、cにおける待ちが解消されたときであるので$wait$、$c.wait$を

$false$ 、$lwset$ をϕにし、また $olwset$ を $olwset$ と $lwset$ の和集合から $c.cont$ を除いた集合としたのち、c と d を結合する。それから、スキップした文の実行を $c.cont$ に格納した接続を用いて再実行し、再び入力チャネルcにおいて待ちが発生する場合を考慮するためにθ'に接続する。それゆえθ'は最小不動点で求められる。

一方、θ_2はcで待ちがないときか、あるいはdが空であるときであり、$lwset$ を $olwset$ と $lwset$ の和集合としたのち、c と d を結合してからθに接続する。$lwset$ を和集合とするのはチャネル結合文の外側にチャネル結合文が存在し、その入力チャネル変数による文がS に含まれた場合を $lwset$ で記憶しおくためである。

$$
\begin{aligned}
&\mathcal{C}[\![\texttt{chan c} \leftarrow \texttt{d}:\ \texttt{S}]\!]_g\rho\theta = \mathcal{C}[\![\ (olwset, lwset) := (\phi,\phi)\,;\texttt{S}\ ]\!]_g\rho\theta' \\
&\quad where \\
&\theta' \ = \ fix(\ \lambda\theta'.\ \mathcal{E}[\![\ c.wait \wedge \neg\texttt{empty(d)}\ ]\!]\rho\texttt{cond}(\theta_1,\theta_2)\ ) \\
&\theta_1 \ = \ \lambda\sigma.\left(\mathcal{C}[\![\ (wait, c.wait, lwset) := (false, false, \phi)\ ;\ \texttt{connect(c} \leftarrow \texttt{d)}\ ]\!]_g\rho\ (\sigma.\rho.c.cont(\theta'))\right)\sigma' \\
&\theta_2 \ = \ \mathcal{C}[\![\ lwset := (olwset \cup lwset)\ ;\ \texttt{connect(c} \leftarrow \texttt{d)}\ ]\!]_g\rho\ \theta \\
&\sigma' \ = \ (\sigma[\sigma.\rho.olwset \cup \sigma.\rho.lwset - \{\rho.c.cont\}/\rho.olwset])
\end{aligned}
$$

$$
\mathcal{C}[\![\ \texttt{connect(c} \leftarrow \texttt{d)}\ ]\!] = \mathcal{C}[\![\ c.buff\ :=\ \texttt{append}\ (\ c.buff\ ,\ \texttt{d}\ )\ ;\ \texttt{d}\ :=\ \epsilon]\!]
$$

7）並列実行文

並列実行文は各文に意味関数を適用し、その結果である状態σ_1、σ_2を合成する。ここで、l_0は未使用のロケーションの集合で各文に適用する際、各文で生成されるローカル変数の記憶領域の衝突を避けるためにロケーションの集合を partition で分割している。

$$
\begin{aligned}
\mathcal{C}[\![\ \texttt{S}_1\texttt{//S}_2\ ]\!]_g\rho\theta\sigma = \ \{\sigma[\ & \sigma_1.\rho.Varlist(\texttt{S}_1)/\rho.Varlist(\texttt{S}_1)\ ,\ \sigma_2.\rho.Varlist(\texttt{S}_2)/\rho.Varlist(\texttt{S}_2)\ , \\
& \sigma_1.\rho.lwset \cup \sigma_2.\rho.lwset/\rho.lwset\ ,\ \sigma_1.\rho.wlabel \cup \sigma_2.\rho.wlabel/\rho.wlabel\ , \\
& \sigma_1.\rho.wait \vee \sigma_2.\rho.wait/\rho.wait\ ,\ \sigma_1.l_0 \cup \sigma_2.l_0/l_0\ ] \\
& \mid \sigma_i \in \mathcal{C}[\![\texttt{S}_i]\!]_g\rho\ I\ (\sigma[l_i/l_0])\quad (i=1,2), I\ =\ \lambda\sigma.\{\sigma\}, \texttt{partition}(\sigma.l_0) = (l_1,l_2)\ \}
\end{aligned}
$$

8）非決定的選択文

非決定的選択文は各文 $\texttt{S}_1$と $\texttt{S}_2$の実行結果の集合の和として与えられる。

$$
\mathcal{C}[\![\ \texttt{S}_1\ []\ \texttt{S}_2\ ]\!]_g\rho\ \theta\sigma = \mathcal{C}[\![\texttt{S}_1]\!]_g\rho\theta\sigma\ \cup\ \mathcal{C}[\![\texttt{S}_2]\!]_g\rho\theta\sigma
$$

9）繰り返し文

繰り返し文は条件式Eが真である間、文S を繰り返し実行する。しかし、文S 中で待ちが発生した場合、待ちを解消するために繰り返し文を脱出する。

$$
\begin{aligned}
&\mathcal{C}[\![\texttt{while}\ E\ \texttt{do}\ \texttt{S}]\!]_g\rho\ \theta = \theta' \\
&\quad where \qquad \theta' \ = \ fix(\ \lambda\theta'.\ \mathcal{E}[\![E]\!]\rho\ \texttt{cond}\ (\ \theta'',\ \theta\ )\ ) \\
&\qquad\qquad\ \ \theta'' \ = \ \mathcal{C}[\![\texttt{S}]\!]_g\rho(\mathcal{E}[\![wait]\!]\rho\ \texttt{cond}\ (\theta,\theta'))
\end{aligned}
$$

10）プロセス呼び出し文

プロセス呼び出し文は、グローバル環境からプロセス変数 p に対する意味関数を受けとり、それにプロセスの引数$\bar{\texttt{c}}$、$\bar{\texttt{x}}$を渡し、ローカル環境ρ、状態σのもとで評価した結果を表す状態σ'のもとで待ちが発生したかどうかを判別する。待ちが発生したとき各入力チャネル変数で本当に待ちが発生しているかを検査し、発生している場合はθ_{11}に、そうでないときにはエラーであるのでθに接続する。θ_{11}は待ちが発生している入力チャネル変数 c_iに対して $c_i.cont$ にそれ以降の接続を格納する。一方、θ_2は待ちが発生していないので$\bar{\texttt{c}}$、$\bar{\texttt{x}}$の内容を更新する。

ここで$\bar{\texttt{c}}$、$\bar{\texttt{x}}$はそれぞれ、入力チャネル変数とそれ以外の変数の並びであり、$\bar{\texttt{c}} = (\texttt{c}_1,\cdots,\texttt{c}_n)$、$\bar{\texttt{x}} = (\texttt{x}_1,\cdots,\texttt{x}_m)$。

$$
\mathcal{C}[\![l:\ \texttt{p}(\bar{\texttt{c}},\bar{\texttt{x}})]\!]_g\rho\ \theta = \lambda\sigma.(\mathcal{E}[\![wait]\!]\rho\ cond\ (\theta_1,\theta_2)\ \sigma')
$$

$$
\begin{aligned}
\theta_1 \ &= \ \mathcal{E}[\![c_1.wait \vee \cdots \vee c_n.wait]\!]\texttt{cond}(\theta_{11},\theta) \\
\theta_2 \ &= \ \lambda\sigma''.\ \theta\ (\sigma[\sigma''.\rho.\bar{\texttt{c}}.buff/\rho.\bar{\texttt{c}}.buff\ ,\ \sigma''.\rho.\bar{\texttt{x}}/\rho.\bar{\texttt{x}}\ ]) \\
\theta_{11} \ &= \ \lambda\sigma''.\ \theta\,(\sigma''[\ \lambda\theta'.\ (\sigma''.\rho.c_i.cont)\,(g(l,c_i)\rho\theta')\,/\rho.c_i.cont,\ \text{for all}\ \ c_i\ \ \text{such that}\ \ c_i.wait = true\ ]\ ) \\
&\quad where \qquad \sigma' \in (g.\sigma.\rho.\texttt{p})(\rho.\bar{\texttt{c}},\rho.\bar{\texttt{x}})\rho I\sigma
\end{aligned}
$$

3.2.7.ラベル接続の意味関数 $\mathcal{L}$ 、$\mathcal{L}_p$、$\mathcal{L}_c$

ラベル接続の意味関数はラベルと入力チャネル変数を用い、プログラムから入力待ちとなり得る文—チャネル

入力文、プロセス呼び出し文—からの接続を構成する関数である。接続の構成法は、チャネル入力文あるいはプロセス呼び出し文がチャネル結合文に囲まれる場合とプロセス呼び出し文によって囲まれる場合に分けられる。例えば、チャネル入力文がプロセス呼び出し文によって囲まれる場合を図4を用いて直感的に説明する。

入力チャネル変数に対して、待ちを解消できるのはその入力チャネルを結合しているチャネル結合文の最後である。したがって、チャネル入力文やプロセス呼び出し文がチャネル結合文によって囲まれている場合は、その文からチャネル結合文の最後までの接続θ_cを構成する。

同様に、プロセス呼び出し文で囲まれている場合においてもその待ちは外側にあるチャネル結合文の最後で解消される。本稿では、チャネル入力文あるいはプロセス呼び出し文からプロセスの最後までの接続をまず構成する。そして、実際に入力待ちが発生したときには、プロセスが呼び出される順序に従って各プロセスに対応する接続を合成することによって、入力待ちが発生した文からチャネル結合文の最後までの接続を構成する方法を採用している。

なお、詳しくは付録参照。

4.おわりに

本稿において非決定的選択、動的なプロセス生成などの基本的な機能を持つ並列処理言語に対して、簡潔で抽象度の高い表示的意味記述を与えるために接続法の概念が有効であることを明らかにした。なお、本研究で与えた意味記述は逐次型処理言語の表示的意味記述の自然な拡張となっている。

本稿で考察した並列処理言語とほぼ同等な機能を持つ言語に対する意味記述が、Broy、Lengauer[8]によって報告されている。しかしながら、その意味記述に関する十分に厳密な議論が不足しているように思われる。

今後の研究課題としては、今回与えた表示的意味$\mathcal{D}$が適切であるかどうかを検討することがある。これは例えば十分に抽象的(fully abstract)であること、すなわち形式的に操作的意味$\mathcal{O}$を与え、任意のプログラムp_1、p_2に対して、$\mathcal{O}(p_1) = \mathcal{O}(p_2) \Rightarrow \mathcal{D}(p_1) = \mathcal{D}(p_2)$が成立することを示すことである。また、今回与えた表示的意味の枠組の中で不動点帰納法などによりプログラムの性質を明らかにし、プログラムの正当性検証への応用を考察することなどが挙げられる。

参考文献

[1] 中島 玲二: 数理情報学入門 スコットプログラム理論; 朝倉書店 (1982)

[2] 那須 隆, 大山口 通夫: プロセス間通信に基づく並列処理言語の表示的意味記述について; 電子情報通信学会春季全国大会(1989), 6.321

[3] 大山口 通夫, 那須 隆: プロセス間通信に基づく並列処理モデルと並列処理言語の表示的意味記述について; 情報関連学会連合大会 (1989), 5.67-70

[4] Milne, R.,Strachey, C.: A Theory of Programming Language Semantics I, II;Champman and Hall (1976)

[5] Stoy, J.: Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory; MIT Press (1977)

[6] G.D.Plotkin: A Power Domain Construction; SIAM J.Compt. Sept.(1976) pp.452-487

[7] M.B.Smyth: Power Domains; JCSS 16 (1978) pp.23-36

[8] M.Broy, C.Lengauer: On Denotational versus Predicative Semantics; technical report of Univ. PASSAU Aug.1987

[9] G.Kahn: The semantics of a simple language for parallel programming; IFIP 74 (1974) pp.471-475

[10] G.Kahn, D.B.MacQueen: "Coroutines and network of parallel processes"; IFIP 77 (1977) pp.993-998

[11] J.D.Brock, W.B.Ackerman: "Scenarios:A Model of Nondeterminate Computation"; LNCS 107 (1981) pp.252-259

[12] de Bruin: The Denotational Semantics Dynamic Networks of Processes; ACM Trans. Prog. Lang. Syst., Oct.(1985)

[13] J.N.Kok: "Denotational semantics of nets with nondeterminism; LNCS 206 (1986) pp.237-249

[14] B.Jonsson: A Fully Abstarct Trace Model for Dataflow Networks; the ACM Sympo. on POPL (1989)

付録.　意味関数

プログラムの意味関数　　(Δ_i :プロセス宣言, Θ :ブロック)

$$\mathcal{P}ry[\![\texttt{program}\Delta_1;\cdots;\Delta_m;\Theta]\!]_g = \mathcal{B}lk[\![\Theta]\!](\mathcal{P}l[\![\texttt{program}\Delta_1;\cdots;\Delta_m;\Theta]\!]_g)$$

プロセス宣言.ラベルの意味関数　　(l_i : ラベル, c_i : inpch)

$$\mathcal{P}l[\![\Psi]\!]_g = g[\gamma_1/\mathtt{P}_1,\ldots,\gamma_m/\mathtt{P}_m,\xi_1/(l_1,c_1),\ldots,\xi_n/(l_n,c_n)]$$

$$(\gamma_1,\ldots,\gamma_m,\xi_1,\ldots,\xi_n) = fix(\ \lambda(\gamma_1,\ldots,\gamma_m,\xi_1,\ldots,\xi_n).\ (\lambda\mu_1\rho_1.\mathcal{B}lk[\![\Theta_1]\!]_{g'}(\rho_1[\mu_1/a_1])),\ \vdots\ (\lambda\mu_m\rho_m.\mathcal{B}lk[\![\Theta_m]\!]_{g'}(\rho_m[\mu_m/a_m])),\ \mathcal{L}(l_1,c_1)[\![\Psi]\!]_{g'},\cdots,\mathcal{L}(l_n,c_n)[\![\Psi]\!]_{g'})$$

where

$$\Psi = \texttt{program}\Delta_1;\cdots;\Delta_m;\Theta$$
$$\Delta_i = \texttt{process}\ \mathtt{P_i}(a_i);\Theta_i \quad 1\le i\le m$$
$$g' = g[\gamma_1/\mathtt{P}_1,\ldots,\gamma_m/\mathtt{P}_m,\xi_1/(l_1,c_1),\ldots,\xi_n/(l_n,c_n)]$$

ブロックの意味関数　　(δ : パラメータ, S : 文)

$$\mathcal{B}lk[\![\texttt{var}\ \delta;\mathtt{S}]\!]_g\rho\theta = \mathcal{D}ec[\![\texttt{var}\ \delta]\!]\rho\,(\lambda\rho'.\mathcal{C}[\![\ \mathtt{S}\ ]\!]_g\rho'\theta\ )$$

変数宣言の意味関数　　(τ : inpch 以外の型)

$$\mathcal{D}ec[\![\texttt{var}\ \mathtt{x}:\tau;\delta]\!]\rho\chi = \mathcal{D}ec[\![\texttt{var}\ \mathtt{x}:\tau]\!]\rho\ (\lambda\rho'.\ \mathcal{D}ec[\![\texttt{var}\ \delta]\!]\rho'\chi\ )$$

$$\mathcal{D}ec[\![\texttt{var}\ \mathtt{x}:\tau]\!]\rho\chi\sigma = (\lambda l_x.\chi(\rho[l_x/\mathtt{x}])\,(\sigma[removeloc(\sigma.l_0,l_x)/l_0,z/l_x]))\,(getloc(\sigma.l_0))$$

$$\mathcal{D}ec[\![\texttt{var}\ \mathtt{c}:\texttt{inpch}]\!]\rho\chi\sigma = \big(\lambda\bar{l}_c.\chi\big(\rho[\bar{l}_c/\bar{c}]\big)\big(\sigma[removeloc(\sigma.l_0,\bar{l}_c)/l_0,\langle false,\epsilon,I\rangle/\bar{l}_c]\big)\big)\,(getloc^3(\sigma.l_0))$$

where　$\bar{c} = \langle c.wait, c.buff, c.cont\rangle$, $\bar{l}_c = \langle l_w, l_b, l_n\rangle$

式の意味関数　　(E_i : 式)

$$\mathcal{E}[\![\ (E_1,\cdots,E_n)\ ]\!]\rho\kappa\sigma = \kappa\ (\langle\mathcal{V}[\![E_1]\!]\rho\sigma,\cdots,\mathcal{V}[\![E_n]\!]\rho\sigma\rangle)\,(\sigma)$$

where

$$\mathcal{V}[\![\mathtt{x}]\!]\rho\sigma = \sigma.\rho.x \quad (\ x\in Id\ )$$
$$\mathcal{V}[\![\ E_1\ \texttt{op}\ E_2\ ]\!]\rho\sigma = (\ \mathcal{V}[\![\ E_1\ ]\!]\rho\sigma\ )\ \texttt{op}\ (\ \mathcal{V}[\![\ E_2\ ]\!]\rho\sigma\ )$$

文の意味関数

(S, $\mathtt{S_i}$: 文, E : 式, c : inpch, d : outch, x : それ以外変数)

$$\mathcal{C}[\![\ \mathtt{S}_1\ ;\ \mathtt{S}_2\ ]\!]_g\rho\,\theta = \mathcal{C}[\mathtt{S}_1]\!]_g\rho(\ \mathcal{E}[\![wait]\!]\rho\ \mathrm{cond}\ (\theta\ ,\ \mathcal{C}[\mathtt{S}_2]\!]_g\rho\,\theta\ )\ )$$

$$\mathcal{C}[\![\texttt{if}\ E\ \texttt{then}\ \mathtt{S}_1\ \texttt{else}\ \mathtt{S}_2]\!]_g\rho\,\theta = \mathcal{E}[\![E]\!]\rho\ \mathrm{cond}\ (\ \mathcal{C}[\mathtt{S}_1]\!]_g\rho\,\theta\ ,\ \mathcal{C}[\mathtt{S}_2]\!]_g\rho\,\theta\ )$$

$$\mathcal{C}[\![(\mathtt{x}_1,\cdots,\mathtt{x}_n) := (E_1,\cdots,E_n)]\!]_g\rho\,\theta = \mathcal{E}[\![(E_1,\cdots,E_n)]\!]_\rho\kappa$$

where

$$\kappa = (\ \lambda\langle\nu_1,\cdots,\nu_n\rangle\sigma.\ \theta(\sigma[\ \langle\nu_1,\cdots,\nu_n\rangle/\langle\rho.\mathtt{x}_1,\cdots,\rho.\mathtt{x}_n\rangle\ ])\ )$$

$$\mathcal{C}[\![l:\ \mathtt{c}\ ?\ \mathtt{x}]\!]_g\rho\,\theta = \mathcal{E}[\![\neg\texttt{empty(c)}]\!]_\rho\ \mathrm{cond}\ (\ \theta'\ ,\ \theta''\ )$$

where

$$\theta' = \mathcal{C}[\![\ (\mathtt{x}, c.buff) := (\texttt{first}(c.buff), \texttt{rest}(c.buff))\ ]\!]_g\rho\,\theta$$
$$\theta'' = \lambda\sigma.\ \mathcal{C}[\![\ (wait, c.wait, wlabel) := (true, true, wlabel\cup\{l\})\ ]\!]_g\ \rho\,\theta\,(\sigma[g(l,\mathtt{c})\rho/\rho.c.cont, \sigma.\rho.lwset\cup\{\rho.c.cont\}/\rho.lwset])$$

$$\mathcal{C}[\![\mathtt{d}\ !\ E]\!]_g\rho\ \theta = \mathcal{E}[\![E]\!]_\rho(\ \lambda\nu\sigma.\ \theta\sigma[\ \mathrm{append}(\sigma.\rho.\mathtt{d},\ \nu)/\rho.\mathtt{d}\ ]\ )\ )$$

$$\mathcal{C}[\![\texttt{chan}\ \mathtt{c}\leftarrow\mathtt{d}:\ \mathtt{S}]\!]_g\rho\,\theta = \mathcal{C}[\![\ (olwset, lwset) := (\phi,\phi)\,;\mathtt{S}\ ]\!]_g\rho\,\theta'$$

where

$$\theta' = fix(\ \lambda\theta'.\ \mathcal{E}[\![\ c.wait\wedge\neg\texttt{empty(d)}\ ]\!]\rho\mathrm{cond}(\theta_1,\theta_2)\ )$$
$$\theta_1 = \lambda\sigma.(\mathcal{C}[\![\ (wait, c.wait, lwset) := (false, false, \phi)\ ;\ \texttt{connect(c}\leftarrow\texttt{d)}\ ]\!]_g\rho\ (\sigma.\rho.c.cont(\theta'))\ )\ \sigma'$$
$$\sigma' = (\sigma[\sigma.\rho.olwset\cup\sigma.\rho.lwset - \{\rho.c.cont\}/\rho.olwset])$$
$$\theta_2 = \mathcal{C}[\![\ lwset := (olwset\cup lwset)\ ;\ \texttt{connect(c}\leftarrow\texttt{d)}\ ]\!]_g\rho\ \theta$$

$$\mathcal{C}[\![\texttt{while}\ E\ \texttt{do}\ \mathtt{S}]\!]_g\rho\ \theta = \theta'$$

where

$$\theta' = fix(\ \lambda\theta'.\ \mathcal{E}[\![E]\!]\rho\ \mathrm{cond}\ (\ \theta''\ ,\ \theta\ )\ )$$
$$\theta'' = \mathcal{C}[\![\mathtt{S}]\!]_g\rho\,(\mathcal{E}[\![wait]\!]\rho\ \mathrm{cond}\ (\theta,\theta'))$$

$$\mathcal{C}[\![\ \mathtt{S}_1\ []\ \mathtt{S}_2\ ]\!]_g\rho\ \theta\sigma = \mathcal{C}[\mathtt{S}_1]\!]_g\rho\theta\sigma\ \cup\ \mathcal{C}[\mathtt{S}_2]\!]_g\rho\theta\sigma$$

$$\mathcal{C}[\![\ \mathtt{S}_1//\ \mathtt{S}_2\ ]\!]_g\rho\theta\sigma = \{\ \sigma[\ \sigma_1.\rho.Varlist(\mathtt{S}_1)/\rho.Varlist(\mathtt{S}_1)\ ,\ \sigma_2.\rho.Varlist(\mathtt{S}_2)/\rho.Varlist(\mathtt{S}_2)\ ,\ \sigma_1.\rho.lwset\cup\sigma_2.\rho.lwset/\rho.lwset\ ,\ \sigma_1.\rho.wlabel\cup\sigma_2.\rho.wlabel/\rho.wlabel\ ,\ \sigma_1.\rho.wait\vee\sigma_2.\rho.wait/\rho.wait\ ,\ \sigma_1.l_0\cup\sigma_2.l_0/l_0\ ]\ |\ \sigma_i\in\mathcal{C}[\mathtt{S_i}]\!]_g\rho\ I\ (\sigma[l_i/l_0])\ \ (i=1,2)\ ,\ I = \lambda\sigma.\{\sigma\}\ \texttt{partition}(\sigma.l_0) = (l_1,l_2)\ \}$$

$$\mathcal{C}[\![l:\ \mathtt{p}(\bar{\mathtt{c}},\bar{\mathtt{x}})]\!]_g\rho\ \theta = \lambda\sigma.(\mathcal{E}[\![wait]\!]\rho\ cond\ (\theta_1,\theta_2)\ \sigma')$$

$$\theta_1 = \mathcal{E}[\![c_1.wait\vee\cdots\vee c_n.wait]\!]\mathrm{cond}(\theta_{11},\theta)$$
$$\theta_2 = \lambda\sigma''.\ \theta\ (\sigma[\sigma''.\rho.\bar{\mathtt{c}}.buff/\rho.\bar{\mathtt{c}}.buff\ ,\ \sigma''.\rho.\bar{\mathtt{x}}/\rho.\bar{\mathtt{x}}\ ])$$
$$\theta_{11} = \lambda\sigma''.\ \theta\,(\sigma''[\ \lambda\theta'.\ (\sigma''.\rho.c_i.cont)\,(g(l,c_i)\rho\theta')/\rho.c_i.cont \text{ , for all } c_i \text{ such that } c_i.wait = true\ ])$$

where　$\sigma'\in(g.\sigma.\rho.\mathtt{p})(\rho.\bar{\mathtt{c}},\rho.\bar{\mathtt{x}})\rho I\sigma$

ラベル接続の意味関数

プログラム

$$\mathcal{L}(l,c)[\![\texttt{program}\Delta_1;\Delta_2;\cdots;\Delta_m;\Theta]\!]_g = \mathrm{cond}\,(\mathcal{L}(l,c)[\![\Delta_1]\!]_g, \mathcal{L}(l,c)[\![\texttt{program}\Delta_2;\cdots;\Delta_m;\Theta]\!]_g) \quad (l\in Label(\Delta_1))$$

プロセス

$$\mathcal{L}(l,c)[\![\texttt{process}\mathtt{P}_i(\bar{\mathtt{c}_i});\Theta_i]\!]_g = \mathrm{cond}\,(\mathcal{L}_p(l,c)[\![\Theta_i]\!]_g, \mathcal{L}_c(l,c)[\![\Theta_i]\!]_g)\,(c\in\bar{\mathtt{c}_i})$$

変数宣言

$$\mathcal{L}_p(l,c)[\![\mathtt{var}\delta;\mathtt{S}]\!]_g = \mathcal{L}_p(l,c)[\![\mathtt{S}]\!]_g$$
$$\mathcal{L}_c(l,c)[\![\mathtt{var}\delta;\mathtt{S}]\!]_g = \mathcal{L}_p(l,c)[\![\texttt{chan c}\leftarrow\mathtt{d}:\ \mathtt{S}']\!]_g$$

(但し、chan c $\leftarrow$ d : S′は S 中の文であり、$l\in Label(\mathtt{S})$とする。)

複文

$$\mathcal{L}_p(l,c)[\![\mathtt{S}_1;\mathtt{S}_2]\!]_g\rho = \mathrm{cond}\,(\lambda\theta.\mathcal{L}_p(l,c)[\![\mathtt{S}_1]\!]_g\rho\,(\mathcal{E}[\![wait]\!]\rho\kappa)\,,\mathcal{L}_p(l,c)[\![\mathtt{S}_2]\!]_g\rho) \quad (l\in Label\,(\mathtt{S}_1))$$

where　$\kappa = \mathrm{cond}(\ \theta\ ,\ \mathcal{C}[\![\mathtt{S}_2]\!]_g\rho\theta\ )$

条件文

$$\mathcal{L}_p(l,c)[\![\texttt{if}\ E\ \texttt{then}\ \mathtt{S}_1\ \texttt{else}\ \mathtt{S}_2]\!]_g = \mathrm{cond}\,(\mathcal{L}_p(l,c)[\![\mathtt{S}_1]\!]_g\ ,\ \mathcal{L}_p(l,c)[\![\mathtt{S}_2]\!]_g)\ \ (l\in Label(S_1))$$

チャネル入力文

$$\mathcal{L}_p(l,c)[\![l:\ \mathtt{c}\ ?\ \mathtt{x}]\!]_g = \mathcal{C}[\![l:\ \mathtt{c}\ ?\ \mathtt{x}]\!]_g$$

チャネル結合文　　($\mathtt{e}\neq\mathtt{c}$)

$$\mathcal{L}_p(l,c)[\![\texttt{chan e}\leftarrow\mathtt{f}:\ \mathtt{S}]\!]_g = \lambda\rho\theta.\mathcal{L}_p(l,c)[\![\mathtt{S}]\!]_g\rho\ \theta'$$

where

$$\theta' = fix(\ \lambda\theta'.\ \mathcal{E}[\![e.wait\wedge\neg\texttt{empty(f)}]\!]\rho\mathrm{cond}(\theta_1,\theta_2)\ )$$
$$\theta_1 = \lambda\sigma.(\mathcal{C}[\![\ (wait, c.wait, lwset) := (false, false, \phi)\ ;\ \texttt{connect(e}\leftarrow\texttt{f)}\ ]\!]_g\rho\ (\sigma.\rho.e.cont(\theta'))\ )\ \sigma'$$
$$\sigma' = (\sigma[\sigma.\rho.olwset\cup\sigma.\rho.lwset - \{\rho.e.cont\}/\rho.olwset])$$
$$\theta_2 = \mathcal{C}[\![\ lwset := (olwset\cup lwset)\ ;\ \texttt{connect(e}\leftarrow\texttt{f)}\ ]\!]_g\rho\ \theta$$

並列実行文

$$\mathcal{L}_p(l,c)[\![\mathtt{S}_1\ //\ \mathtt{S}_2]\!]_g\rho\theta = \mathrm{cond}(\ \theta_1\ ,\ \theta_2\ )\ (l\in Label(\mathtt{S}_1))$$

where

$$\theta_i = \mathcal{L}_p(l,c)[\![\mathtt{S}_i]\!]_g\rho\,(\mathcal{C}[\![\texttt{waitset}(Label(\mathtt{S}_1)\cup Label(\mathtt{S}_2))]\!]_g\rho\theta)$$
$$\mathcal{C}[\![\texttt{waitset}(Label)]\!]_g\rho\theta\sigma = \theta\,(\sigma[\sigma.\rho.wlabel\cap Label\neq\phi/\rho.wait]) \quad (\text{但し, } i=1,2)$$

非決定的選択文

$$\mathcal{L}_p(l,c)[\![\ \mathtt{S}_1\ []\ \mathtt{S}_2\ ]\!]_g = \mathrm{cond}\,(\mathcal{L}_p(l,c)[\![\mathtt{S}_1]\!]_g, \mathcal{L}_p(l,c)[\![\mathtt{S}_2]\!]_g) \quad (l\in Label(\mathtt{S}_1))$$

繰り返し文

$$\mathcal{L}_p(l,c)[\![\texttt{while}\ E\ \texttt{do}\ \mathtt{S}]\!]_g = \lambda\rho\theta.\mathcal{L}_p(l,c)[\![\mathtt{S}]\!]_g\rho\theta'$$

where　$\theta' = \mathcal{E}[\![wait]\!]\rho\mathrm{cond}\,(\theta,\mathcal{C}[\![\texttt{while}\ E\ \texttt{do}\ \mathtt{S}]\!]_g\rho\theta)$

プロセス呼び出し文

$$\mathcal{L}_p(l,c)[\![l:\ \mathtt{p}(\bar{\mathtt{c}},\bar{\mathtt{x}})]\!]_g = \lambda\rho\theta.\mathcal{E}[\![c_1.wait\vee\cdots\vee c_n.wait]\!]\rho\kappa$$

where

$$\kappa = \mathrm{cond}(\theta_1\ ,\ \theta)$$
$$\theta_1 = \mathcal{C}[\![wait\ :=\ true]\!]_g\rho\ (\lambda\sigma.\theta\,(\sigma[\lambda\theta'.(\sigma.\rho.c_i.cont)(g(l,c_i)\rho\theta')/\rho.c_i.cont\ ,\ \text{for all } c_i \text{ such that } \rho.c_i.cont\in lwset])\ )$$

なお、その他の文 Sに対しては $\mathcal{L}_c(l,c)[\![\mathtt{S}]\!]_g = undef$ である。