分散環境に置かれたGHCプログラムの意味論のための意味領域

加藤 暢[†] 村上昌己[†]

現在の計算機ネットワーク環境において，複数のプログラムが各々異なった計算機上で，互いに通信しながら実行されることはもはや日常的なことである．こういった環境ではプログラムを分割して分散環境に配置することも可能である．このとき1つの計算機上に置かれた完結したプログラムだけでは適切な回答が得られないが，ネットワークにつながった他の計算機上のプログラムからの情報を用いれば適切な回答が得られる場合がある．このような分散環境においては，完結したプログラムとしては等しい2つのプログラムが，他の計算機上のプログラムと通信しながら動作するときには異なる動作をするような場合が存在する．本論文ではこれらのプログラムが並行論理型言語GHCである場合を想定し，これらのプログラムを区別できるような意味論を構成するための意味領域を提案する．プログラムの動作はある半順序集合によって表現される．さらにプログラムの実行の枝別れを表現するために，入れ子になった {} と () によって構成された構造を導入する．

A Domain for a Semantics of Guarded Horn Clauses for Programs on Distributed Environment

TORU KATO[†] and MASAKI MURAKAMI[†]

Recent network technology made programs that are allocated in dispersed computers possible to communicate each other. We can split a program into some fragments and distribute them to dispersed computers on a network. In such environment, we sometimes get a proper result which cannot be got from a program fragment running on a local computer by using another program fragment running on a remote computer. In such case, two program fragments that are identical as complete program act as different programs when they communicate another programs on such a computer network. Thus these two programs must be distinguished. We present a domain for a semantics of GHC programs which can discriminate such two programs above. The behavior of programs are represented by a collection of partially ordered events. We introduce a structure constructed from nested pairs of {} and ().

1. はじめに

現在のネットワーク環境において，複数のプログラムが各々異なった計算機上で，互いに通信しながら実行されることはもはや日常的なことである．こういった環境ではプログラムを分割して分散環境に配置することも可能である．このとき，1つの計算機上に置かれた完結したプログラムだけでは適切な回答が得られないが，ネットワークにつながった他の計算機上のプログラムからの情報を用いれば適切な回答が得られる場合がある．

これらのプログラムが並行論理型言語によるプログラムである場合，上のような現象は以下のように具体的に説明できる．次のような実行環境が実現されている場合を考える：あるプログラムにゴールが与えられると，通常の実行系と同様，プログラム内からコミットできる節を探す．しかし他のプログラムにもコミットできる可能性のある節が存在すると分かっている場合は，ネットワークを通じて他のプログラムにもそのゴールが送られコミットできる節を探す．そして他のプログラム内の節がコミット可能なら，元のプログラム内の節と同様に非決定的に選択され，計算が進行する．

このような実行環境においては，個々のプログラムのみではデッドロックあるいは失敗としか見なせない計算も，ネットワーク全体で見ると単なるサスペンドであり，他のプログラムの節情報が届いたとたんに正常な計算として適当な回答が得られる場合がある．

このような状況下では，個々のプログラム単独では

† 岡山大学工学部
Faculty of Engineering, Okayama University

互いにまったく同じ振舞いをする2つのプログラムが，他のプログラムの節情報を用いると，各々異なる振舞いをする場合がある．

並行論理型言語に対してはこれまでに数多くの意味論が提案されてきた[3),4),6),8),9),11)]．また我々は文献7)においてOR合成可能な意味論を提案した．この意味論を用いると，上記のようなネットワークを通じて通信するプログラムの動作もある程度は表現できるようになる．

しかしこれらの意味論では，あるプログラムに意味を与えるとき，他のプログラムとの通信によって生じる動作の違いを表現できない場合があるか，できたとしても非常に制約の強いものになっている．

例1.1 次の3つのGHCプログラム V, U, T について考える：

$$
\begin{aligned}
V = \{\ & p(X,Y) :- X = [a] | true, q(Y,Z), t(Z).\\
& p(X,Y) :- X = [d] | Y = [d].\\
& q(Y,Z) :- Z = [b] | Y = [b].\\
& q(Y,Z) :- Z = [c] | Y = [c].\\
& s(Y,Z) :- Z = [b] | Y = [b].\\
& r(Y,Z) :- Z = [c] | Y = [c].\}.
\end{aligned}
$$

$$
\begin{aligned}
U = \{\ & p(X,Y) :- X = [a] | true, s(Y,Z), t(Z).\\
& p(X,Y) :- X = [a] | true, r(Y,Z), t(Z).\\
& p(X,Y) :- X = [d] | Y = [d].\\
& q(Y,Z) :- Z = [b] | Y = [b].\\
& q(Y,Z) :- Z = [c] | Y = [c].\\
& s(Y,Z) :- Z = [b] | Y = [b].\\
& r(Y,Z) :- Z = [c] | Y = [c].\}.
\end{aligned}
$$

$$
T = \{\ t(Z) :- true | Z = [b].\ \}.
$$

たとえば文献7)では，述語 p の動作を表現する意味領域上の要素は V, U ともに次のようになる：

$$
\begin{aligned}
&\mathcal{O}_\Omega(V) = \mathcal{O}_\Omega(U)\\
&= \{\ p(X,Y) :- \{\ \langle X = [a] | true\rangle,\\
&\qquad\qquad \langle X = [a], Z = [b] | Y = [b]\rangle,\\
&\qquad\qquad \langle X = [a], t(Z) | true\rangle\},\\
&\qquad p(X,Y) :- \{\ \langle X = [a] | true\rangle,\\
&\qquad\qquad \langle X = [a], Z = [c] | Y = [c]\rangle,\\
&\qquad\qquad \langle X = [a], t(Z) | true\rangle\},\\
&\qquad p(X,Y) :- \{\ \langle X = [d] | Y = [d]\rangle\ \},\\
&\qquad \vdots\\
&\}.
\end{aligned}
$$

そのため V と U を区別できない．ただし $\mathcal{O}_\Omega$ は文献7)で定義された意味関数である．直観的に上記集合の要素は，ゴール $?-p(X,Y)$ に対する各々のプログラムの実行過程を表したものである．$\langle X = [a], Z = [b] | Y = [b]\rangle$ は，X と Z が各々 $[a]$ と $[b]$ へと具体化されると，Y を $[b]$ に具体化することが可能である，という事象を表している．また $\langle X = [a], t(Z) | true\rangle$ は X が $[a]$ へと具体化された後，ゴール $t(Z)$ を呼び出すという事象を表している．これらはGHCのコミット規則[12)]を反映したものである．

また入出力の系列を木構造を持たない要素で表現する他の意味論でもやはり同様に V と U を区別できない．ところがネットワークを通じてプログラム T の節も使用できる場合，ゴール $?-p([a],Y)$ に対して，プログラム V では失敗する計算は存在しないが U には存在する．これはゴール $q(Y,Z)$ と $s(Y,Z)$ に対するプログラムの動作が，ゴール $t(Z)$ に対する動作とは矛盾しないが，$r(Y,Z)$ と $t(Z)$ では動作に矛盾が生じるためである．したがって V と U のようなプログラムを区別できるようにプログラムの意味を構成してやる必要がある． □

意味領域上の要素に木構造を持たせるというものがある[5)]．文献5)で行われたようにプログラムの動作を木構造を持った要素で表すと，V と U を区別できるようになる．ところで文献5)では並行に走るプロセス全体の動作を個々の動作のinterleaveによって表している．しかし並行プロセスの動作のinterleaveではプロセスの持つ潜在的な並行度を陽に表せない．

プロセスの持つ潜在的な並行度を陽に表すというtrue concurrencyの立場に立ち，さらに V と U のようなプログラムを区別できるようにするためには，プログラム実行中に現れるすべての選択可能な実行経路をAnd-Or木で表現する必要がある．しかし単純なAnd-Or木で表現するだけでは新たな問題が生じる．あるプログラムの実行中にとりうるすべての選択可能な動作を表す巨大なAnd-Or木を考える．その中の入出力動作を表す任意の要素間の関係，つまりそれらの要素が互いに枝別れのない1本の実行経路の上にあるのか，あるいは枝別れしている別の実行経路上にあるのかは，その巨大な木をたどり，さらにそれらの要素どうしを比較しなければ分からないのである．

プログラムの構文を直接解析することに比べ，プログラムの解析を容易にするということは意味論の持つべき利点の1つである．しかし上記のように木をたどらなければプログラムの解析ができないのでは，その意味論がこの利点を持っているとは言い難い．

そこで本論文では例1.1の V と U を区別でき，また実行中に現れる入出力動作どうしの関係を容易に解析できるように，プログラムの動作を，{} と () の入れ子で構成された枝別れを持った構造体からなる半順

序集合を用いて表現する．

2. Ω 開プログラム

本章では本稿で扱う GHC Ω 開プログラムを定義する．Ω 開プログラムとは文献 2) において論理型言語に対し OR 合成可能な意味論を提案するために導入されたものである．意味論の OR 合成可能性とはプログラムの和集合演算に対して意味論が合成可能性を保つことである[2),7)]．

GHC Ω 開プログラムとは，文献 2) における Ω 開プログラムの概念を GHC プログラムに適用したものである．

定義 2.1 本論文では GHC[12)] のサブセットである Flat GHC[11)] を議論の対象とする．本論文で扱う GHC プログラム P とは以下のような Flat GHC 節の有限集合である：

$$h{:-}\sigma|U_1,\ldots,U_n,b_1,\ldots,b_k.$$

ただし h は互いに異なる変数を引数として持つアトム，σ は単一化ゴールの集合，各 U_i は単一化ゴール，各 b_i は単一化以外の述語記号を持つアトムを表す．さらに本論文で扱うプログラムには変数のリネーミングによって互いに等しくなるような節は存在しないものとする． □

定義 2.2 （関数 **Pred**） X を任意の GHC プログラム，またはプログラム節，またはアトムの集合，またはアトムとする．**Pred**(X) は X 中に現れるすべてのアトムの述語記号からなる集合である． □

定義 2.3 P をプログラム，Ω を述語記号の集合とする．P 中に以下のような節が存在するとき，P を Ω 開プログラムという．

$$h{:-}\sigma|U_1,\ldots,U_n,b_1,\ldots,b_n.$$
$$\text{ただし } \exists m(1\leq m\leq n).\mathbf{Pred}(b_m)\subseteq\Omega. \quad \square$$

直観的に Ω とは他のプログラムによって定義が拡張されうる述語の述語記号を集めたものである．そして Ω 開プログラムとは，プログラムの OR 合成によって新たな計算の可能性が生じる節を持つようなプログラムのことである．定義 2.3 にある節はこのようなものである．

3. 入出力履歴

本稿では文献 11) で提案された入出力履歴を拡張して用い，プログラムに意味を与える．本章では本稿で用いる入出力履歴と，それを用いたプログラムの意味領域の形式的な定義を与える．

GHC プログラムに対し，文献 11) では入出力履歴と呼ばれる要素の集合によって意味を与えた．入出力履歴とはプログラムの操作的意味との健全性を保ち，また AND 合成可能性を満たすために必要とされる，プロセスの入出力動作の系列を記述した要素である[10)]．また文献 7) では意味論の OR 合成可能性を満たすために，この入出力履歴に特別な形をした要素を持つことを許した．

本稿では例 1.1 における U，V のようなプログラムを区別できるように，入出力履歴をさらに拡張し，{} と () の入れ子で構成された枝別れを持った構造体となった入出力履歴によってプログラムに意味を与える．

Var を変数の集合とする．そして *Fun* を関数記号の集合とし *Fun* の要素で arity 0 のものを定数とする．項を関数記号と変数から通常どおり定義する[1)]．さらに項から項へのマッピングである代入も通常どおり定義する[1)]．

定義 3.1 項 τ が $\tau\in Var$，または定数，または arity n の関数記号 f と n 個の相異なる変数 $X_1,\ldots,X_n$ に対して $\tau=f(X_1,\ldots X_n)$ という形をしているとき，τ を単項と呼ぶ． □

定義 3.2 τ を単項，$X\in Var$ とする．このとき $X=\tau$ を単一化ゴールと呼ぶ．単一化ゴールとは，単一化と呼ばれる述語記号 "=" を持つアトムである．単一化ゴール $X=X$ を *true* と記述する． □

単一化ゴールの有限集合は，そのすべての元を実行して成功したときに得られる代入を定義している．たとえば $\{X=[A|Y],A=a\}$ という単一化ゴールの集合は A を a に，X を $[a|Y]$ に写像する代入と見なすことができる[11)]．単一化ゴールの有限集合をそれが定める代入と同一視する．すなわち単一化ゴールの集合の間の等式（$\sigma_1=\sigma_2$ など）は，代入としての等しさを表す．

定義 3.3 （関係 $\models$） σ を単一化ゴールの集合，U を単一化ゴールとする．$\sigma\cup\{U\}$ が代入を定義するとき「U は σ に矛盾しない」という．さらに $\sigma\cup\{U\}$ が σ とまったく同じ代入になっているとき

$$\sigma\models U$$

と記述する． □

定義 3.4 ガード付き単一化[11)]とは，代入 σ と，ゴール U の対 $\langle\sigma|U\rangle$ または代入 σ および単一化ゴール以外のゴールアトムのタプル $\tilde{b}$ とゴール *true* の対 $\langle\sigma,\tilde{b}|true\rangle$ である．ただし **Pred**$(\tilde{b})$ は Ω の部分集合である． □

直観的に $\langle\sigma|U\rangle$ は，「ゴールの引数が σ のように具体化されると U という単一化を実行する」という事象を表している．

また $\langle \sigma, \tilde{b} | true \rangle$ はゴールの引数が σ のように具体化されると実行が進んで $\tilde{b}$ というゴールを呼び出すという事象を表している．

定義 3.5 ガード付き単一化 $\langle \sigma | U \rangle$ または $\langle \sigma, \tilde{b} | true \rangle$ に対して σ を入力代入，U と $true$ を出力代入と呼ぶ．ガード付き単一化 gu に対して $\mathbf{In}(gu)$ は gu の入力代入を，$\mathbf{Out}(gu)$ は gu の出力代入を表すものとする． □

ガード付き単一化には次のような順序関係が定義される．

定義 3.6 （関係 $\prec$） gu_1, gu_2 をガード付き単一化とする．

$$gu_1 \prec gu_2 \quad \text{iff} \quad \exists\theta_1.\mathbf{In}(gu_1)\theta_1 = \mathbf{In}(gu_2) \wedge \forall\theta_2.\mathbf{In}(gu_1) \neq \mathbf{In}(gu_2)\theta_2.$$

ただし $\mathbf{In}(gu_1)\theta$ は代入 θ と $\mathbf{In}(gu_1)$ の合成によって得られる代入を表す． □

直観的に $\langle \sigma_1 | U_1 \rangle \prec \langle \sigma_2 | U_2 \rangle$ とは，U_1 よりも U_2 の方が実行するまでにより多くの具体化を必要とすることを表している．つまり U_2 が実行可能となった時点では，U_1 もすでに実行可能となっている．

定義 3.7 GU をガード付き単一化の集合，Gu を GU の有限部分集合とする．Gu は以下の条件を満たすとき，またそのときにかぎり下向きに閉じているという．

$gu' \prec gu$ を満たす GU 中の任意のガード付き単一化 gu, gu' に対し

$$gu \in Gu \rightarrow gu' \in Gu.$$ □

定義 3.8 （フロンティア） ガード付き単一化の集合 GU の下向きに閉じている部分集合 Gu を考える．任意の $gu'(\in Gu)$ に対し，GU の要素 gu が

$$gu \not\prec gu' \text{ かつ } gu' \neq gu$$

を満たすとき，gu を GU に関する Gu のフロンティアと呼ぶ．また GU 中に $gu'' \prec gu$ を満たすような Gu のフロンティア gu'' が存在しないとき，フロンティア gu を GU に関する Gu の極小フロンティアと呼ぶ． □

GU が，あるゴールの実行によって起きるすべての事象の集合であったとき，GU の下向きに閉じている部分集合 Gu とは，計算のある途中の時点までに起こる事象の集合を表している．GU に関する Gu のフロンティアとは，Gu によって表される計算が終わった時点より未来に起こる GU 内の事象を表す．極小フロンティアとは，この計算が Gu まで進んだ時点の直後に行われうる GU 内の事象を表している．

定義 3.9 仮ガード付きストリームを次のように帰納的に定義する：

(1) 1個以上たかだか加算無限個のガード付き単一化からなる集合は仮ガード付きストリームである．

(2) GU を (1) の条件を満たす仮ガード付きストリームとする．また G を仮ガード付きストリームのたかだか加算無限集合とする．このとき，$GU \cup \{G\}$ は仮ガード付きストリームである． □

本稿では以後仮ガード付きストリームを他の集合と区別するために，ある集合が上記の仮ガード付きストリームの条件を満たすとき，またそのときに限り {} の代わりに () で要素を囲む．またある**条件**を満たす**要素**からなる集合が仮ガード付きストリームになるとき，一般の集合同様（**要素** | **条件**）という記法を用いる．

定義 3.10 （二項関係 $\underset{\text{tree}}{\in}$, $\underset{\text{strm}}{\in}$） G, G' を仮ガード付きストリームの集合，GU, GU' を仮ガード付きストリーム，gu をガード付き単一化とする．二項関係 $\underset{\text{strm}}{\in}$, $\underset{\text{tree}}{\in}$ を次のように帰納的に定義する（ただし二項関係式の右辺が仮ガード付きストリームのときのみ $\underset{\text{strm}}{\in}$ を，仮ガード付きストリームの集合のときのみ $\underset{\text{tree}}{\in}$ を定義する）：

- $gu \underset{\text{strm}}{\in} GU$ if $gu \in GU$.
  $gu \underset{\text{strm}}{\in} GU$ if $\exists G.G \in GU, gu \underset{\text{tree}}{\in} G$.
  $G \underset{\text{strm}}{\in} GU$ if $G \in GU$.
  $G \underset{\text{strm}}{\in} GU$ if $\exists G'.G' \in GU, G \underset{\text{tree}}{\in} G'$.
  $GU' \underset{\text{strm}}{\in} GU$ if $\exists G'.G' \in GU, GU' \underset{\text{tree}}{\in} G'$.
- $GU \underset{\text{tree}}{\in} G$ if $GU \in G$.
  $GU \underset{\text{tree}}{\in} G$ if $\exists GU' \in G, GU \underset{\text{strm}}{\in} GU'$
  $gu \underset{\text{tree}}{\in} G$ if $\exists GU.GU \in G, gu \underset{\text{strm}}{\in} GU$.
  $G' \underset{\text{tree}}{\in} G$ if $\exists GU.GU \in G, G \underset{\text{strm}}{\in} GU$.

□

直観的に $\underset{\text{strm}}{\in}$ は，集合とその要素の関係 $\in$ を，仮ガード付きストリームとその中に現れるすべての要素との関係を表すことができるように拡張したものである．また $\underset{\text{tree}}{\in}$ は，仮ガード付きストリームの集合とその中に現れるすべての要素との関係を表すことができるように，$\in$ を拡張したものである．

例 3.11 $G = \{GU_1, GU_2\}$ を仮ガード付きストリームの集合とする．また

$$GU_1 = (gu_1, gu_2, \{(gu_1^1, gu_1^2), GU_1^2\}),$$

とする．このとき gu_1^1 に対して $\underset{\text{strm}}{\in}$ の2行目の定義より $gu_1^1 \underset{\text{strm}}{\in} GU_1$ となる．また $\underset{\text{tree}}{\in}$ の3行目の定義より $gu_1^1 \underset{\text{strm}}{\in} G$ となる．また $\underset{\text{tree}}{\in}$ の4行目の定義より $\{(gu_1^1, gu_1^2), GU_1^2\} \underset{\text{tree}}{\in} G$ となる． □

定義 3.12（関数 **path, Path**） G を仮ガード付きストリームの集合とする．関数 **Path, path** を以下のように定義する：

$$\mathbf{Path}(G) = \bigcup_{GU \in G} \mathbf{path}(GU).$$

ただしガード付き単一化の集合 Gu と仮ガード付きストリームの集合 G' を用いて $GU = Gu \cup \{G'\}$ と表すとき，

$$\mathbf{path}(GU) = \{Gu \cup GU' | GU' \in \mathbf{Path}(G')\}.$$

$\mathbf{Path}(G)$ 中の任意の要素を GU とするとき，$\{GU\}$ を G のパスという．また $\mathbf{path}(GU)$ 中の任意の要素 GU' を GU のパスという． □

例 3.13 $G = \{GU_1, GU_2\}$ を仮ガード付きストリームの集合とする．また

$$\begin{aligned} &GU_1 \\ &\quad = (gu_1, gu_2, \{\ (gu_1^1, gu_1^2), \\ &\qquad\qquad (gu_2^1, gu_2^2, \{\ (gu_3^1, gu_3^2), \\ &\qquad\qquad\qquad (gu_4^1, gu_4^2)\}\})\} \end{aligned}$$

とする．このとき

$$\begin{aligned} &\mathbf{path}(GU_1) \\ &\quad = \{\ (gu_1, gu_2, gu_1^1, gu_1^2), \\ &\qquad (gu_1, gu_2, gu_2^1, gu_2^2, gu_3^1, gu_3^2), \\ &\qquad (gu_1, gu_2, gu_2^1, gu_2^2, gu_4^1, gu_4^2)\} \end{aligned}$$

となる．

$(gu_1, gu_2, gu_1^1, gu_1^2), (gu_1, gu_2, gu_2^1, gu_2^2, gu_3^1, gu_3^2)$ 等，$\mathbf{path}(GU_1)$ の任意の要素は GU_1 のパスである．また $Gu_2 \in \mathbf{path}(GU_2)$ とするとき，これら GU_1, GU_2 の任意のパスに対して

$$\begin{aligned} &\{(gu_1, gu_2, gu_1^1, gu_1^2)\}, \\ &\{(gu_1, gu_2, gu_2^1, gu_2^2, gu_3^1, gu_3^2)\}, \\ &\{Gu_2\}, \ldots \end{aligned}$$

等が各々 G のパスである． □

直観的に パス とは，複数の枝別れを持つ木を根からたどっていったときの，1 つの経路のことである．そして **Path** および **path** はそれらの経路をすべて集めるためのものである．

定義 3.14（関数 $|\cdot|$） ガード付き単一化 $\langle\sigma|U\rangle$ に対して，単一化ゴールの集合 $|\langle\sigma|U\rangle|$ を以下のように定義する：

$$|\langle\sigma|U\rangle| = \sigma \cup \{U\}.$$

ガード付き単一化 $\langle\sigma, \widetilde{b}|true\rangle$ に対して，単一化ゴールの集合 $|\langle\sigma, \widetilde{b}|true\rangle|$ を以下のように定義する：

$$|\langle\sigma, \widetilde{b}|true\rangle| = \sigma.$$

GU を仮ガード付きストリームとするとき，$|GU|$ とは以下のような単一化ゴールの集合である：

$$|GU| = \bigcup_{gu \underset{\text{strm}}{\in} GU} |gu|.$$ □

定義 3.15（ガード付きストリーム） GU を仮ガード付きストリームとし，$GU_1 \in \mathbf{path}(GU)$ とする．そしてガード付き単一化の集合 Gu を次のような条件を満たすものとする：

$$\begin{aligned} &\forall gu \in Gu.gu \in GU_1 \\ &\wedge(\forall gu' \in GU_1.gu' \prec gu \rightarrow gu' \in Gu). \end{aligned}$$

この条件を満たすすべての Gu と，GU_1 に関する Gu の任意の極小フロンティア $\langle\sigma|U\rangle$ に対して

U は $|Gu| \cup \sigma$ と矛盾せず，

任意の $U' \in \sigma$ に対して，U' は $|Gu|$ と矛盾せず，

$|Gu| = \sigma\theta$ を満たすような θ は存在しない．

この条件がすべての $GU_1 \in \mathbf{path}(GU)$ に対して成り立つとき，GU をガード付きストリームと呼ぶ． □

直観的に仮ガード付きストリームとは OR 分岐による複数の経路を 1 つの要素で表現したものである．ガード付きストリームとはすべての経路において，その経路を構成するガード付き単一化の集合が矛盾なく GHC プログラムの実際の計算を表現しているような仮ガード付きストリームのことである．

例 3.16 まったく同じガード付き単一化からなる 2 つの仮ガード付きストリーム GU_1, GU_2 を以下のようなものとする：

$$\begin{aligned} &GU_1 \\ &\quad = (\langle X = [a] | true\rangle, \{\ (\langle X = [a] | Y = [b]\rangle), \\ &\qquad\qquad (\langle X = [a] | Y = [c]\rangle)\}). \\ &GU_2 \\ &\quad = (\langle X = [a] | true\rangle, \langle X = [a] | Y = [b]\rangle, \\ &\qquad \langle X = [a] | Y = [c]\rangle). \end{aligned}$$

このとき，

$$\begin{aligned} &\mathbf{path}(GU_1) \\ &\quad = \{\ (\langle X = [a] | true\rangle, \langle X = [a] | Y = [b]\rangle), \\ &\qquad (\langle X = [a] | true\rangle, \langle X = [a] | Y = [c]\rangle)\} \end{aligned}$$

であり GU_1 のすべてのパスは定義 3.15 の条件を満たすのでガード付きストリームである．一方 $\mathbf{path}(GU_2) = \{GU_2\}$ であり，GU_2 自身が GU_2 の唯一のパスとなる．定義 3.15 中に現れる Gu の 1 つは $\{\langle X = [a] | Y = [c]\rangle\}$ であり，また $\langle X = [a] | Y = [b]\rangle$ は GU_2 に関する Gu の極小フロンティアの 1 つである．$Y = [b]$ は $|Gu|$ と矛盾するので GU_2 はガード付き単一化ではない． □

上の例に現れるガード付き単一化 $\langle X=[a]|Y=[b]\rangle$ は $\langle\{X=[a]\}|Y=[b]\rangle$ を省略して表したものである．また $\langle\{X=[a]\},t(Z)|true\rangle$ という形をしたガード付き単一化は $\langle X=[a],t(Z)|true\rangle$ と省略して記述する．以下では簡単化のため，この省略記法を用いる．

定義 3.17 ガード付きストリームの集合をガード付きストリーム木という． □

本論文では以後明示的なことわりがない限り，gu または gu に添字を付けた記号でガード付き単一化を，GU または GU に添字を付けた記号でガード付きストリームを，G または G に添字を付けた記号でガード付きストリーム木を表すものとする．

本論文では1個以上たかだか加算無限個のガード付き単一化とたかだか1個のガード付きストリーム木よりなる集合を () で囲んで表し，たかだか加算無限個のガード付きストリームよりなる集合を {} で囲んで表す．こうすることによりこれら2種類の集合を明確に区別する．

同じ実行系列上に存在し，並行または逐次的に実行されうるアトミックな入出力動作どうしは () で囲まれた集合の要素として表される．一方各々別の実行系列上に存在し，決して両方ともが実行されることはないアトミックな入出力動作どうしは {} で囲まれた集合中の各々別の () の中に記述する．つまり同じガード付ストリーム木上に存在するある2つのアトミックな入出力動作を表すガード付き単一化 gu_1，gu_2 を見たとき，それらの関係は次のように考えてよい．もし gu_1，gu_2 を同時に囲む最も内側の括弧が () なら，gu_1，gu_2 はある1つの実行系列上の入出力動作である．もし gu_1，gu_2 を同時に囲む最も内側の括弧が {} ならば，gu_1，gu_2 は別の実行系列上に存在し，一方が実行されれば，もう一方は実行されることはない．

このように本稿では () と {} に特別な意味を持たせるため，ガード付きストリーム（木）に対する各種演算の区切りを明示するときには [ ] を用いる．

例 3.18 例 1.1 で定義したプログラム V の述語 $p(X,Y)$ に対する入出力履歴を例にとって上に述べた括弧の役割を説明する．本稿で提案する意味領域上では $p(X,Y)$ の動作は次の入出力履歴で表される：

$$
\begin{aligned}
p(X,Y):-\{(&\langle X=[a]|true\rangle,\\
&\{(\langle X=[a],Z=[b]|Y=[b]\rangle,\\
&\ \langle X=[a],t(Z)|true\rangle),\\
&\ (\langle X=[a],Z=[c]|Y=[c]\rangle,\\
&\ \langle X=[a],t(Z)|true\rangle)\}),\\
&(\langle X=[d]|Y=[d]\rangle)\}.
\end{aligned}
$$

$\langle X=[a]|true\rangle$ という入出力動作と $\langle X=[a],Z=[b]|Y=[b]\rangle$ という入出力動作は，これらを同時に囲む最も内側の括弧が () なので同一実行系列上で並行に実行されうる．一方 $\langle X=[a],Z=[b]||Y=[b]\rangle$ と $\langle X=[a],Z=[c]|Y=[c]\rangle$ はこれらを同時に囲む最も内側の括弧が {} なので別々の系列上での入出力動作であり，並行に実行されることはない．

このように機械的に括弧の入れ子関係を調べるだけで，ガード付きストリーム木上の任意の入出力動作どうしの関係が分かるのである． □

例 3.19 以下の式 (1) は単純な And-Or 木を用いて，あるプログラムの動作を表現したものである．ただし × を And 分岐，+ を Or 分岐を表す記号とする．αとβ をプロセスの動作を表す要素とするとき，$\alpha\times\beta$ は α という動作と β という動作が並行または逐次的に実行されることを表す．また $\alpha+\beta$ は α または β のどちらか一方が実行されることを表す．また A，A'，B，B' はアトミックな入出力動作を表す要素とする．

$$(\ldots,(A+A'))\times(\ldots,(B+B')). \qquad (1)$$

これは **図1** のような木を表したものである．ただし ',' は逐次動作を表している．

式 (1) において，A' と B が同一実行系列上に属し並行または逐次的に実行されうるのか，あるいは別々の実行系列上に属するのかは，木をたどり，さらに A' と B が相矛盾する入出力動作であるのかないのかを確かめなければ分からない．たとえば $A=\langle X=[a]|Y=[a]\rangle$，$A'=\langle X=[b]|Y=[b]\rangle$，$B=\langle X=[a]|Z=[a]\rangle$，$B'=\langle X=[b]|Z=[b]\rangle$ であった場合，A' と B は別々の実行系列上にのみ存在する．しかしその判定のためにはまず (1) の木をたどり，(1) 上での A' と B の位置関係を確かめた後，A' の入力代入 $X=[b]$ と B の入力代入 $X=[a]$ とが矛盾することを調べなければならない．

一方本論文で提案するガード付きストリームでは図1に表されるようなプログラムの動作は次のように

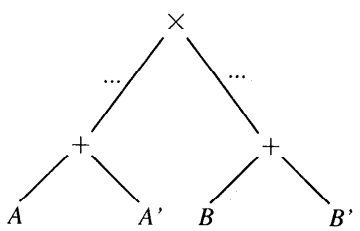

図1 式 (1) の概念図
Fig. 1 An illustration of Eq. (1).

表現される：

$$(\{(\ldots,A,\ldots,B),(\ldots,A',\ldots,B')\}).$$

したがって例 3.18 と同様の方法で A, A', B, B' 各々の関係は容易に判定できるのである．

なお

$$(\{\ (\ldots,A,\ldots,B),(\ldots,A',\ldots,B'),\ (\ldots,A',\ldots,B)\}),$$

という仮ガード付きストリームはガード付きストリームの条件（定義 3.15）を満たさないので，このようなガード付きストリームは存在しない．つまり A' と B が並行または逐次的に実行されるという誤った情報は存在できないのである． □

定義 3.20（**仮入出力履歴**） 仮入出力履歴とは，プロセスが起動されたときの形を表すヘッド h と，そのプロセスが実行可能なすべての入出力の履歴を木構造で表したガード付きストリーム木 G を用いて次のように表される：

h:−G.

ここで h は互いに異なる変数を引数として持つアトムである． □

仮入出力履歴 $t = h$:−G に対して h を t のヘッド部，G を t のボディー部と呼ぶ．

定義 3.21（**入出力履歴，履歴基底，IP**） すべての仮入出力履歴からなる集合上で，変数のリネーミングによって定まる同値関係によってできる商集合を履歴基底と呼ぶ．また履歴基底の要素を入出力履歴と呼ぶ．履歴基底の冪集合の部分集合で以下の条件を満たすものを IP とする：

$$IP = \{I \mid I \in \text{履歴基底の冪集合かつ } \forall t,t' \in I.\ (\mathbf{head}(t) = \mathbf{head}(t') \rightarrow t = t')\}.$$

□

直観的に IP とは，同じヘッドを持つ入出力履歴を複数個持たない集合からなる集合である．このため IP の要素は，比較可能な入出力履歴を複数個持たないような集合となる．

定義 3.22（**二項関係 $\leq$**） 入出力履歴間の二項関係 $\leq$ を以下のように定義する：同じヘッドを持つ入出力履歴 h:−G_1, h:−G_2 に対して

h:−$G_1 \leq h$:−G_2 iff $G_1 \subseteq G_2$. □

定義 3.23（**二項関係 $\sqsubseteq$**） IP 上の二項関係 $\sqsubseteq$ を次のように定義する：IP の任意の要素 I_1, I_2 に対して

$I_1 \sqsubseteq I_2$ iff $\forall t_1 \in I_1, \exists t_2 \in I_2.t_1 \leq t_2$. □

J を定義 3.24 に現れる D 中のすべての要素に付けられたインデックスよりなる集合とする．以降で現れる $\bigcap G_j$ は $\bigcap_{j\in J} G_j$ の省略形である．

定義 3.24（**演算 $\bigsqcap_{IP}$**） IP の任意の部分集合 $D = \{I_1, I_2, \ldots\}$ に対して，入出力履歴の集合 $\bigsqcap_{IP} D$ を次のように定義する：

$$\bigsqcap_{IP} D = \{h\text{:−}G_D \mid \forall I_j \in D \exists h\text{:−}G_j \in I_j.\ G_D = \bigcap G_j\}.$$

□

定理 3.25 $D = \{I_1, I_2, \ldots\}$ を IP の任意の部分集合とする．$\bigsqcap_{IP} D$ は D の最大下界である． □

証明（**下界性**） $\bigsqcap_{IP} D$ 中の任意の要素 h:−$\bigcap G_j$ に対して，定義 3.24 より D 中のすべての I_j の中には h:−G_j が存在する．$\bigcap G_j \subseteq G_j$ は明らかなので h:−$\bigcap G_j \leq h$:−G_j となり，$\bigsqcap_{IP} D \sqsubseteq I_j$ が成立する．

（**最大性**） D の任意の下界を I とする．定義 3.23 より次の関係が成り立っている：

$$\forall I_j \in D \forall t \in I \exists t_j \in I_j.t \leq t_j.$$

上式に現れる t, t_j を各々 h:−G, h:−G_j とすると，定義 3.22 より任意の j について $G \subseteq G_j$ となっている．したがって $G \subseteq \bigcap G_j$ である．h:−$\bigcap G_j$ なるガード付きストリームは定義 3.24 より $\bigsqcap_{IP} D$ の要素として存在するので，定義 3.23 より $I \sqsubseteq \bigsqcap_{IP} D$ となる．

定理 3.26 IP は $\sqsubseteq$ を順序として持つ cpo である．以下のような入出力履歴の集合 $I_\top$ が IP の最大の要素である：

$$I_\top = \{h\text{:−}G \mid h \text{ は互いに異なる変数を引数として持つ述語かつ } G = \bigcup_{h\text{:−}G' \in \text{履歴基底}} G'\}.$$ □

証明 順序 $\sqsubseteq$ が反射律，推移律を満たすことは定義 3.22, 3.23 より明らか．通常定義 3.23 のような順序は反対称律が成り立たないため前順序にしかならない．以下では IP の性質により反対称律が成り立つことを示す．

IP の要素 I_1, I_2 に対し，$I_1 \sqsubseteq I_2$, $I_2 \sqsubseteq I_1$ が成り立っているとする．$I_1 \sqsubseteq I_2$ より次の関係が成り立っている：

$$\forall t_1 \in I_1, \exists t_2 \in I_2.t_1 \leq t_2. \quad (2)$$

一方 $I_2 \sqsubseteq I_1$ より $\forall t_2' \in I_2, \exists t_1' \in I_1.t_2' \leq t_1'$ が成り立っているので，式 (2) に現れる t_2 に対しても t_1' が I_1 中に存在し，次の関係が成り立っている：

$$t_2 \leq t_1'. \quad (3)$$

ところで定義 3.22 より $t_2 \leq t_1'$ であるためには t_1' と t_2 は同じヘッドを持っていなければならない．同様に t_1 と t_2 も同じヘッドを持っている．定義 3.21 より IP の要素は同じヘッドを持つ入出力履歴を複数個持たない．したがって I_1 中の入出力履歴である t_1 と t_1' は同じものでなければならない．したがって式 (2), (3) より $t_1 \leq t_2 \wedge t_2 \leq t_1$ となる．定義 3.22 より入出力履歴 t_1 と t_2 の順序はそれらのボディー部の包含関係によって決まり，包含関係に関しては反対称律が成立するので $t_1 = t_2$ となる．

以上より I_1 中の任意の要素 t に対して $t \in I_2$ であり，逆も成り立つので $I_1 = I_2$ となる．よって $(IP, \sqsubseteq)$ は半順序集合となる．

IP 中の任意の要素 I に対して $I \sqsubseteq I_\top$ は定義 3.23 より明らか．

定理 3.25 より IP の任意の部分集合 D に対して最大下界 $\bigsqcap_{IP} D$ が存在し，$\bigsqcap_{IP} D$ は，その作り方より IP の要素であることは明らか． □

$(IP, \sqsubseteq)$ が cpo であることが証明できたので，もし $(IP, \sqsubseteq)$ 上に適当な連続関数が定義できれば，GHC Ω 開プログラムを IP のある要素へ写像できる最大不動点意味論が構成できる．

例 3.27 例 1.1 のプログラム V の述語 $p(X,Y)$ の動作を表す入出力履歴は例 3.18 で述べたものであり，プログラム U の述語 p に対する入出力履歴は次のようになる：

$$
\begin{aligned}
p(X,Y) :- \{\ (\ & \langle X = [a] | true \rangle, \\
& \langle X = [a], Z = [b] | Y = [b] \rangle, \\
& \langle X = [a], t(Z) | true \rangle), \\
(\ & \langle X = [a] | true \rangle, \\
& \langle X = [a], Z = [c] | Y = [c] \rangle, \\
& \langle X = [a], t(Z) | true \rangle), \\
(\ & \langle X = [d] | Y = [d] \rangle\ )\}.
\end{aligned}
$$

したがってプログラム U と V の区別が可能となっていることが分かる． □

この例に示したように，プログラムの構造が異なるために動作が異なる可能性があるにもかかわらず，文献 7) では区別できなかった 2 つのプログラムを本意味論では区別できるのである．文献 5) ではこれらを区別するために I/O アクションの Interleaving による手法を用いていた．本論文では Non-interleaving な手法を用いたために，「何と何が同時に起こるのか」という解析が容易にできるのである．

ところで，例 3.27 に示した入出力履歴中にガード付き単一化 $\langle X = [a], t(Z) | true \rangle$ が存在することで，他のプログラム中で定義されている述語 $t(Z)$ に対する節を用いると，新たな動作が生じる可能性のあることを示している[7]．このときの動作の違いを $p(X,Y)$ に対する 2 つの入出力履歴は表している．この $\langle X = [a], t(Z) | true \rangle$ という形をしたガード付き単一化は文献 7) で定義されたものである．文献 7) ではこのガード付き単一化を，プログラムの OR 合成によって生じる動作の違いを検知するために用いている．ネットワークを通じた複数のプログラムの動作を表現することは，プログラムの OR 合成という概念を拡張したものだと見ることができる．したがって本論文で示した意味論は，文献 7) で定義された意味論の自然な拡張になっているのである．つまり，文献 7) の意味論で区別可能であった任意の 2 つのプログラムは，やはり本意味論によっても区別されるのである．

4. 結論・今後の課題

本論文では分散環境に置かれた GHC プログラムに対して意味論を構成するための意味領域の提案を行った．

個々のプログラム単独ではまったく同じ振舞いをするプログラムが，分散環境におかれ互いにネットワークで通信しながら計算を進める場合，互いに異なる振舞いをする場合がある．これまでの意味論ではこういったプログラムどうしの区別が不可能か，可能であったとしても非常に制約の強いものであった．

本論文では {} と () の入れ子で構成された枝別れを持った構造体からなる半順序集合で意味領域を構成することにより，こういったプログラムを区別することが可能となった．この構造体は一種の木構造と見ることができる．

木構造を用いた意味論はこれまでにも提案されている[5]．しかし文献 5) は true concurrency の立場に立ったものではなかった．true concurrency の考えに基づき，さらに上で述べたプログラムどうしを区別できるようにするためには，And-Or 木を用いてプログラムの動作を表現する必要がある．しかし単に入出力動作の親子関係を And-Or 木で表すだけでは，どの入出力動作どうしが並行に実行されうるのか，あるいは別の実行系列上に存在し同時に実行されることがないのかの判別が困難になる．

本論文で提案した構造体を用いると，機械的に括弧の入れ子関係を調べるだけで，すべての実行系列上の任意の入出力動作どうしの関係が分かるのである．単純な And-Or 木を用いるよりも，本論文で提案した構造体を用いる方がこのような解析が容易になっている

ことを例 3.18, 3.19 において示した.

本論文ではまた, この構造体を持った要素からなる領域を用いても従来と同様の最大不動点意味論を構成できるよう, この領域が cpo であることを証明した.

今後の課題は, cpo であるこの領域上での連続関数を定義し, 最大不動点意味論を構成することで, GHC Ω 開プログラムをこの意味領域に形式的に写像できるようにすることである.

ところで, ネットワーク上に分散されたプログラムを部品プログラムと見なし, ネットワーク上のプログラム全体を 1 つの大きなプログラムとして見たとき, この大きなプログラムは分散された部品プログラムを OR 合成したものと見なせる. もし本論文で提案した意味領域を用いた意味論が OR 合成可能性[2)]を満たすものならば, 部品プログラムの意味からネットワーク上のプログラム全体の動作を合成できる. この場合 1 つの部品プログラムを, 安全に他の効率の良いプログラムに交換できる. ここで安全にとは, 部品プログラムを交換することによって, ネットワーク上のプログラム全体が予期せぬ動作を起こさないことが保証されるということである.

そこで構文領域上の OR 合成演算に対応し, 合成可能性を満たすような演算を, 本論文で提案した領域上に構成し, この領域を用いた意味論が OR 合成可能性を満たすことを証明することも課題となる.

参 考 文 献

1) 有川, 原口:述語論理と論理プログラミング, オーム社 (1988).
2) Bossi, A., Gabbrielli, M., Levi, G. and Meo, M.C.: A Compositional Semantics for Logic Prgrams, *Theoretical Computer Science*, 122, pp.3–47 (1994).
3) Falaschi, M., Levi, G., Martelli, M. and Palamidessi, C.: A New Declarative Semantics for Logic Languages, *Proc. 5th Conf. and Symp*, pp.993–1005 (1988).
4) Gabbrielli, M. and Levi, G.: Unfolding Reactive Semantics for Concurrent Constraint Programs, *Lecture Notes in Computer Science*, 463, pp.204–216 (1990).
5) Gabbrielli, M. and Levi, G.: Unfolding and Fixpoint Semantics of Concurrent Constraint logic Programs, *Theoretical Computer Science*, 105, pp.85–128 (1992).
6) Gerth, R., Codish, M., Lichtenstein, Y. and Shapiro, E.: Fully Abstract Denotational Semantics for Flat Concurrent Prolog, *Proc. North American Conf. on Logic Programming*, pp.553–569 (1989).
7) Kato, T. and Murakami, M.: An OR-Compositional Semantics of Guarded Horn Clauses for Programs with Perpetual Processes, *Trans. Information Processing Society of Japan*, Vol.37, No.8, pp.1497–1505 (1996).
8) Levi, G. and Palamidessi, C.: Contributions to the Semantics of Logic Perpetual Processes, *Acta Informatica 25*, pp.691–711 (1988).
9) Maher, M.: Logic Semantics for a Class of Commited Choice Programming, *Proc. 4th Int. Conf. on Logic Programming*, pp.858–876, MIT Press (1987).
10) 村上昌己:並行論理型言語の形式的意味論, 情報処理, Vol.32, No.7, pp.819–838 (1991).
11) Murakami, M.: A Declarative Semantics of Flat Guarded Horn Clauses for Programs with Perpetual Processes, *Theoretical Computer Science*, 75, pp.67–83 (1990).
12) Ueda, K.: Guarded Horn Clauses, Technical Report, TR-103, ICOT (1985).

(平成 8 年 6 月 10 日受付)
(平成 9 年 5 月 8 日採録)

加藤 暢 (学生会員)

1991 年岡山大学工学部情報工学科卒業. 1993 年同大学院工学研究科修士課程修了. 現在同大学院自然科学研究科博士課程在籍. 並行論理型言語の意味論の研究に従事. 日本ソフトウェア科学会会員.

村上 昌己 (正会員)

1980 年名古屋工業大学情報工学科卒業. 1982 年名古屋大学大学院修士課程情報工学修了. 1985 年同博士課程満了. 同年 9 月工学博士. 同年 4 月富士通 (株) 入社. 同年 6 月より 1989 年 7 月まで新世代コンピュータ技術開発機構に出向. 帰社後富士通国際情報社会科学研究所に勤務. 1991 年 4 月より岡山大学工学部助教授. 並列性の理論, プログラムの意味論, プログラム変換/検証等の研究に従事. 電子情報通信学会, 日本ソフトウェア科学会, EATCS 各会員.