ソフトウェア基礎論　３３－６
プログラミング言語　２３－６
（1989. 12. 8）

無限プロセスを含むGHCプログラムの並列導出手続きに基づく操作的意味論

Concurrent Operational Semantics of GHC programs with Perpetual Processes

村上 昌己
Masaki Murakami

富士通国際情報社会科学研究所

International Institute for Advanced Study of
Social Information Science, FUJITSU Ltd.

概要

筆者は先に [Murakami 88] で無限プロセスを含むGHCプログラムの宣言的 / 不動点的セマンティクスについて報告した。本稿では、GHCプログラムの操作的意味論を導入する。ここで提案するセマンティクスは、トップダウン並列導出の手続きを基にしたものであり、無限プロセスを含むGHCプログラムのセマンティクスについて、従来の逐次的な状態遷移モデルを用いた定式化ではなく、先に提案した半順序構造による宣言的なセマンティクスと同じ枠組みを用いて与えられている。これによって、操作的意味論と宣言的意味論の関係について、直接的な対応で議論することが可能となった。

Abstract: A declarative/fixpoint semantics of GHC programs with perpetual processes was reported in [Murakami 88]. In this paper, an operational semantics of GHC programs is reported. The semantics reported here is not based on a sequential state-transition model but on a concurrent top down derivation procedure and uses the partial order model that is similar to the declarative semantics.

1　はじめに

近年、GHC [Ueda 88a] をはじめとするHorn論理に基づく並列プログラミング言語の研究が様々な方面から進められている。これらの言語は、並列プロセスの記述をandで結ばれたゴールの並びによって、またストリームによるプロセス間通信の機能を単一化による変数の具体化によって記述するなど、論理型プログラムの枠組みを用いて、並列プログラミングの基本的な要素を記述している。しかしながら、プロセス間の同期 / 制御については、ガード / コミットの機構を用いて記述しているため、純粋な論理型プログラムの枠組みをはみだしたものとなっている。筆者は先に [Murakami 88] でGHCプログラムのモデル論的セマンティクスについて報告した。そこで提案したセマンティクスは [Lloyd 84] における純Horn論理型プログラムに対する宣言的意味論の拡張であり、これによってガード／コミット機構によって制御されながら無限に計算を続けるプロセスを含むプログラムの性質の議論が可能となった。

しかしながらそこで提案されたセマンティクスは、プログラムの意味をcompositionalに記述するために、実際の計算規則とは直接対応がつかない形で定義されていた。したがって、このようなセマンティクスが直観的にGHCの計算規則と整合性のある、「正しい」セマンティクスであることが理解されるためには、多くの報告で採用されているように、適当な手続き的意味論を与えて、それとの整合性があることが示されることが有力であるものと考えられる。先に報告した意味論が論理型プログラムでいうところのモデル論的なものであったのに対し、手続き的意味論は導出原理の変形拡張からなっており、この意味においてある種の証明論的なものに対応するものと考えることができる。したがってモデル論的意味論と手続き的意味論との整合性（等価性）は、手続き的意味論の健全性 / 完全性という形で議論することが自然であるものと考えられる。

本稿では、健全性 / 完全性の議論ができるようなGHCプログラムの操作的意味論として、並列トップダウン導出の手続きに基づく意味論を導入する。

従来報告されている、並列論理型プログラムの意味論について、操作的意味論と宣言的 / 不動点的意味論あるいは表示的意味論との関係を議論しているものとして

は、[Falaschi 88, Shibayama 87] が挙げられるが、これらはいずれも無限プロセスの動作を true concurrent なモデルを用いて議論することはしていない。

本稿では、不動点的 / 宣言的意味論と操作的意味論との最も大きな違いは次のような点にあると考える。すなわち、先に述べたようにモデル論的 / 不動点的セマンティクスでおいては、プログラムの意味は計算可能なものではあるが、その具体的な計算手順を示す必要はない。したがって、無限プロセスの意味は一般に無限系列あるいは無限半順序集合によって与えられる。これによっていくつかの無限プロセスが並列に走るプログラムのセマンティクスについて、各プロセスのセマンティクスから全体のセマンティクスを compositional に定めることが可能となる。

一方操作的意味論においては、プログラムの意味は具体的な計算規則を実行した結果によって定義される。したがって、無限プロセスのセマンティクスを記述するためには、計算規則の有限回の適用によって得られる有限集合の列の極限によって定められる。 また、操作的意味論においては、ある無限プロセスの結果を、そのプロセスと並列に走るプロセスが参照する場合、無限な計算の結果をできあがってしまったものとして扱うことはできない。したがって、並列に走るプロセスの計算は実行中の他のプロセスとの通信についても同時に記述されなければならない。

本稿では操作的意味論として並列トップダウン導出に基づく計算規則を示す。ここで提案する操作的意味論は、[Saraswat 87] にあるような逐次的な非決定性状態遷移モデルによるものではなく、並列無限導出の手続きによって記述する。並列無限導出の記述のためには、プロセスの動的な生成 / 消滅、導出木を表現するデータ構造の記述方法が与えられなければならない。また先に述べたように並列に走るプロセスについて、各プロセスの実行中に行なわれる途中結果の通信について記述できなければならない。

本稿ではまず、準備として宣言的 / 不動点的意味論及び操作的意味論の両方で用いられる基本的な概念について説明する。次に、並列無限導出の手続きを記述する言語について述べる。次に、GHC のプログラムを用いた並列トップダウン無限導出の手続きを与える。

2 準備

本章では、宣言的 / 不動点的意味論及び操作的意味論の両方で用いられる基本的な概念について説明する。ここで議論するセマンティクスは、いずれも論理型プログラムのセマンティクスを記述する際に用いられた概念を拡張したものを用いて説明される。通常の論理型プログラムの場合、そのセマンティクスはプログラムで成功する基底単位節の集合によって定義される。しかしながら、GHC のような言語の場合、特に無限プロセスのセマンティクスを記述する場合、成功する基底単位節によってはプログラムのセマンティクスを十分に記述することはできない [Takeuchi 86, Murakami 88]。ここでは、GHC における基底単位節にあたる概念として入出力履歴 [Murakami 88] の概念を用いる。入出力履歴は [Saraswat 87] によって定義された senerio の概念に近いものである。しかしながら、入出力履歴は、半順序の構造をもっており、interleaving なモデルではないという点で異なっている。

2.1 入出力履歴

入出力履歴とは次のような形をしたものである。

$$H :- GU$$

ここで H は互いに異なる変数に述語記号を適用したもの, GU はある節にコミットするまでに通過するガードを解くのに必要な代入 σ とコミットした節のボディ部での単一化の実行を表す式 U_b の対 $<\sigma|U_b>$ の集合で次に述べる定義 3 の条件を充たすものである。 直観的には H という形のゴールの引数が σ というガードが解かれるために十分具体化されると、 U_b というボディ部分に出現する単一化ゴールが実行されることを意味する。入出力履歴はプログラムが失敗もデッドロックもなく実行が続いたときの、計算のパスのインスタンスを半順序構造によって表現している。

Var を可算個の変数の集合、 *Fun* を関数記号の有限集合とする。 *Fun*, *Var* からつくられる項の集合 *Terms* は通常のように定義されるものとする。

$\tau \in Terms$、 $X \in Var$ とするとき、 $X = \tau$ という形の式を代入式とよぶ。代入式は GHC プログラムの中に出現したときは、単一化ゴールとして実行されるものである。逆に代入式は以下に定義するセマンティクスの世界に出現したときは、何らかの GHC プログラムに出現する単一化ゴールの実行のうち、左辺の変数を新たに具体化するような実行を表現している。一方、プログラムのボディ部分に出現する単一化ゴールの実行のうち、それまでの計算によって得られている束縛以上に何も具体化ないようなテストユニフィケーションとして実行されたときは、 $X ?= \tau$ のように記述する。このような式をテスト単一化式 (以下では単にテスト式) とよぶ。

$X_1, \ldots, X_n \in Var$ を互いに異なる変数、また $1 \leq j \leq n$ について $X_j \neq \tau_j$ であるとする。このとき代入式の有限集合: $\sigma = \{X_1 = \tau_1, \ldots, X_n = \tau_n\}$ は、その各元を単一化ゴールとして並列に実行したときに得られる代入を定義する。(もちろん任意の代入式の集合が代入を定義するわけではない。) 以下では混乱がない限り、代入式の集合とそれが定義する代入を同一視する。 σ を代入を定義している代入式の集合、 U を代入式とする。 $\sigma \cup \{U\}$ もやはり代入を定義するならば、 U は σ と無矛盾であるという。さらに $\sigma \cup \{U\}$ で定義される代入が σ によって定義される代入と等しいならば、 σ は U

を含意するといい、$\sigma \models U$ で表わす。また、σ が U を含意しないとき、$\sigma \not\models U$ で表わす。

定義 1. (ガード付き単一化: [Murakami 88])

$X \in Var, \tau \in Terms$, σ を代入を定義するような代入式の集合、$< \sigma|uni(X,\tau) >$ をガード付単一化と呼ぶ。ここで、$uni(X,\tau)$ は、$X = \tau$ 又は $X ?= \tau$ を表わす。以下では、σ を $< \sigma|uni(X,\tau) >$ のガード部分、$uni(X,\tau)$ をそのボディ部分とよぶ。

$< \sigma|U >$ をガード付単一化とする。このとき $|< \sigma|U >|$ は以下のように定まる代入式及びテスト式の集合である。

$$|< \sigma|U >| = \{U\} \cup \sigma$$

定義 2. (ガード付代入の集合上の半順序: [Murakami 88])

GU をガード付代入の集合とする。$< \sigma_1|u_1 >, < \sigma_2|u_2 > \in GU$ について、ある代入 θ_1 が存在し、$\theta_1\sigma_1 = \sigma_2$ となり、かつ $\sigma_1 = \theta_2\sigma_2$ となる θ_2 が存在しないとき、

$$< \sigma_1|u_1 > \prec < \sigma_2|u_2 >$$

であると定義する。

$\prec$ が半整列順序であることは容易に示せる。

W を集合、$\leq$ を W の上の半整列順序、W' を W の有限部分集合とする。任意の $w, w' \in W$ について、$w' \leq w$ かつ $w \in W'$ であるならば $w' \in W'$ となるとき、W は下から閉じているという。W' を W の下から閉じた有限部分集合とする。$w \in W$ が任意の $w' \in W'$ について $w' \leq w$ であるならば、w は W' の上界であるという。W' の上界 w が極小であるとは、いかなる $w'' \in W$ についても、w'' が W' の上界でかつ $w'' \leq w$ とはならないことをいう。

GU をガード付単一化の集合、Gu を GU の有限部分集合とする。$\{X = \tau | < \sigma|uni(X,\tau) > \in Gu$ 又は $< \{\ldots, X = \tau, \ldots\}|U > \in Gu\}$ と定義される代入式の集合 (及びそれによって定義される代入) を $|Gu|$ であらわす。

定義 3. (ガード付ストリーム: [Murakami 88])

ガード付単一化の集合 GU が、GU の任意の下から閉じた有限部分集合 Gu 及び Gu の任意の極小上界 $< \sigma|U >$ について以下の条件を満たすとき、ガード付ストリームと呼ぶ。

1) U が代入式であった場合、U は $|Gu| \cup \sigma$ と無矛盾でありかつ $|Gu| \cup \sigma \not\models U$ となる。

2) U がテスト式であった場合、U は $|Gu| \cup \sigma$ と無矛盾である。一方、$|GU| \cup \sigma \models U$ であるならば、U はテスト式である。

3) 任意の $U' \in \sigma$ について、U' は $|Gu|$ と無矛盾であり、かつ $|Gu| = \theta\sigma$ となる代入 θ は存在しない。

次に同期付マージとよばれる演算を定義する。これは並列に走る複数のゴールを含むゴール節の計算を表現するガード付ストリームを、各プロセスの計算を表現するガード付ストリームから合成する演算である。基本的なアイデアは次の通りである。$GU_1, \ldots, GU_n$ をガード付ストリームとする。各 GU_i は i 番目のプロセスの計算を表現しているものとする。$GU_1, \ldots, GU_n$ の同期付マージ ($GU_1 || \cdots || GU_n$ と表記する。) は $GU_1, \ldots, GU_n$ の集合和を基に以下に述べるような操作を加えて得られるものである。すなわち、$< \sigma_i \mid X = \tau > \in GU_i$, $< \sigma_j \mid U > \in GU_j, (X = \tau) \in \sigma_j$ であると仮定する。言い換えれば、i 番目のプロセスが X のプロデューサーであり、j 番目のプロセスが X のコンシューマであるとする。このとき $GU_1 || \cdots || GU_n$ では $< \sigma_j \mid U >$ は $< \sigma_i \cup \sigma_j - \{X = \tau\} \mid U >$ によって置き替えられる。直観的には、このような二つのプロセスが並列に実行された時には、ストリーム X を通じて τ が送られるという通信はこれらのプロセス系内部の通信としてローカライズされ、X に τ が外部から来るのを待っているということは記述されなくなる。

形式的には次の様に定義される。

定義 4. (同期付マージ: [Murakami 88]) $GU_1, \ldots, GU_n$ をガード付ストリームとする。$Gu_k (1 \leq k)$ を以下の様に定義する。

$$\begin{aligned}
Gu_0 = \{ & < \sigma|U > | \exists i, < \sigma|U > \in GU_i, \\
& \forall U' \in \sigma, ((\exists Gu \subset \cup_j GU_j, |Gu| \models U') \Rightarrow \\
& (|Gu| \setminus \{X = \tau | < \sigma''|X = \tau > \in Gu\} \models U'))\}
\end{aligned}$$

$$\begin{aligned}
Gu_{k+1} = Gu_k \cup \{ & < \sigma|U > | \\
& \exists \sigma'((\exists i, (< \sigma'|U > \in GU_i) \wedge \\
& (\forall U' \in \sigma'(\exists Gu \subset Gu_k, |Gu| \models U') \vee \\
& (\forall Gu \subset \cup_j GU_j, \\
& (|Gu| \models U' \Rightarrow \\
& (|Gu| \setminus \{X = \tau | < \sigma''|X = \tau >\} \models U')))) \wedge \\
& (\sigma = (\sigma' \setminus \{U' | |Gu_k| \models U'\}) \cup \\
& \{U'' | \exists Gu \subset Gu_k \\
& (\exists U' \in \sigma'((|Gu| \models U') \wedge \\
& (\forall \theta \subset |Gu|, |Gu| \neq \theta \Rightarrow \theta \not\models U'))) \wedge \\
& (\exists < \sigma''|U''' > \in Gu, U'' \in \sigma'')\})\}
\end{aligned}$$

又 GU を次の様に定義する。

$$GU = \bigcup_{k \to \infty} Gu_k$$

GU がガード付ストリームとなり、かつ

$$\{U | < \sigma|U > \in GU\} = \{U | \exists i < \sigma|U > \in GU_i\}$$

を $GU_1, \ldots, GU_n$ のガード付マージとよび、

$$GU_1 \| \ldots \| GU_n$$

で表記する。

$n = 1$ の場合同期付マージは常に定義され、その結果は GU_1 自身に等しくなる。

直観的には、Gu_k は次のように考えられる。今、$< \sigma | U > \in Gu_k$ であるとする。これは σ が $GU_1 \| \ldots \| GU_n$ の環境から与えられると、高々 k 回の GU_j 同士での通信の後に U が実行可能となることを意味する。例えば、ある $< \sigma | U > \in GU_i$ について、もし $< \sigma | U > \in Gu_0$ であるならば、U は σ を $GU_1 \| \ldots \| GU_n$ の外部から待っていて、かつ $GU_j (j \neq i)$ からは何も待っていないことを意味する。$< \sigma | U > \in Gu_{k+1}$ であった場合は、U は $GU_1 \| \ldots \| GU_n$ の環境からある入力を待ち、なおかつ既に $\sigma_h (1 \leq h \leq m)$ を待っていることがわかっている $Ub_1, \ldots, Ub_m$ がある GU_j で実行されるのを待っている。もし、Ub_h が $U' \in \sigma_h$ が環境から来るのを待っているならば、U も U' を待っている。

定義 5. (↓ 演算: [Murakami 88])

GU をガード付ストリーム、V を変数の有限集合とする。ここで GU の V による制限 $GU \downarrow V$ とは次のような集合である。

$$GU \downarrow V = \{ < \sigma | uni(X, \tau) > | \\ \exists k < \sigma | uni(X, \tau) > \in GU, X \in V_k \}$$

ここで、

$$V_0 = V$$
$$V_{i+1} = V_i \cup \{ X | \exists gu \in GU, \exists uni(Y, \tau) \in |gu|, \\ X \text{は} \tau \text{に含まれ} Y \in V_i, \text{かつ} \\ \forall gu' \in GU, gu' \prec\!< \sigma | U > \text{ならば} \\ X \text{は} gu' \text{に現れない.} \}$$

GU がガード付ストリームのとき、$GU \downarrow V$ もガード付ストリームとなる。

定義 6. (⋈ 演算: [Murakami 88])

GU をガード付ストリーム，θ_1, θ_2 を単純な代入式の集合とする。θ_1, θ_2 と GU から定まる集合 $GU \bowtie (\theta_1, \theta_2)$ は、それがやはりガード付ストリームとなるとき、以下のように定義される。

$$GU \bowtie (\theta_1, \theta_2) =$$
$$\{ < \sigma | U_b > | < \sigma' | U'_b > \in GU, \sigma = \sigma' \cup \theta_1 - \theta_2,$$
$$\text{もし } (\theta_1 \cup \theta_2) \not\models U_b \text{ ならば } U'_b = U_b,$$
$$\text{また}(U_b \text{ が } X = \tau \text{ の形で、かつ}$$
$$(\theta_1 \cup \theta_2) \models U_b$$
$$\text{であるならば } U'_b \text{ は } X ?= \tau. \}$$

次に、先に述べたガード付ストリームの概念を基に、純 Horn 論理型プログラムにおける基底単位節にあたるものを GHC に対して導入する。

ここで以下で扱う GHC プログラムのシンタックスを以下のように定義する。

定義 7. (ガード付節)

述語記号の集合を $Pred$ とする。$H, B_1, B_2, \ldots, B_n$ を $Pred$, $Terms$ から作られる原子式，かつ H の引数部に出現するのは全て異なる変数，$U_{g1}, \ldots, U_{gm}, U_{b1}, \ldots, U_{bh}$ を単純な代入式とする。このとき次の節：

$$H :- U_{g1}, \ldots, U_{gm} | U_{b1}, \ldots, U_{bh}, B_1, B_2, \ldots, B_n$$

をガード付節とよぶ。ガード付節の有限集合をプログラムとよぶ。

以下では，H が $p(X_1, X_2, \ldots, X_k)$ のとき

$$Var(H) = \{ X_1, X_2, \ldots, X_k \}$$

とする。

定義 8. (ゴール)

p を arity k の $Pred$ の元、$X_1, X_2, \ldots, X_k$ を互いに異なる変数、σ を ω 代入とするとき、$\sigma p(X_1, X_2, \ldots, X_k)$ はゴールである。

単一化はいきなりゴールしては現われないことに注意されたい。

定義 9. (ゴール節)

ゴール節とは、ゴール $g_i (1 \leq i \leq n)$ の並び $g_1, \ldots, g_n$ である。

定義 10. (入出力履歴) GU をガード付きストリームとする。入出力履歴 t とは次のようなものである。

$$p(X_1, X_2, \ldots, X_k) : - GU$$

ここで $p \in Pred$ でアリティ k、かつ $X_1, X_2, \ldots, X_k$ は互いに異なる変数であり、

$$GU \downarrow Var(p(X_1, X_2, \ldots, X_k)) = GU.$$

ここで $p(X_1, X_2, \ldots, X_k)$ を t のヘッド部分，GU をボディ部分とよぶ。

入出力履歴は並列言語における単位節の概念に相当する。しかしながら入出力履歴では本質的に同じ計算に対して複数の記法が可能となる。すなわちヘッドに出現する変数以外の変数については，いかなる変数が用いられようと外から観測する限りにおいては同じ計算を表現しているものとみなすことができる。そこで厳密には入出力履歴の領域に変数名のつけかえによる適当な同値関係[1]を導入し、その同値類で Herbrand 基底にあたるものを定義するのが適切である。

[1] [Murakami 88] でこのような同値関係の形式的な定義とその定義の妥当性について議論している。

以下では、*Fun*, *Var*, *Pred* から定まるすべての入出力履歴の集合の変数名のつけかえによる同値類から、適当な代表元を選びだした集合を $I/O - hist$ で表し、その元を入出力履歴とよぶ。

3 操作的意味論

本節では、GHC の操作的意味論を提案する。ここで提案する操作的意味論はトップダウンの並列導出を基にしたもので、与えられたプログラムと実行環境のもとでの与えられたゴール節のトレースに対応するガード付単一化の集合をつくるものである。この手続きは、通常の論理型言語の操作的意味論の場合と同様に、与えられたゴールの導出木をつくりながら処理を進める。通常の純 Horn 論理プログラムの場合、導出木の各節点には基底単位節がラベルとして対応していた。ここで導出木の各節点には、GHC における基底単位節の概念にあたる入出力履歴が対応するものと考えられる。また、ここで提案するセマンティクスは GHC プログラムの無限プロセスの動作を記述することが目的であった。したがって、ここで提案される操作的意味論は、無限集合をボディ部にもつ入出力履歴をラベルにもつ無限導出木を作り続ける操作と考えられる。

3.1 並列導出の記述言語

並列導出の手続きはそれ自身、ある種の簡単な並列アルゴリズムの記述言語によって記述される。ここではまず、この並列言語について簡単に述べる。ここで操作的意味論を記述するために並列言語に必要とされる機能には、次のようなものがある。

並列プロセスの記述: ここで考える導出手続きは、複数のゴールからなるゴール節を、与えられたプログラムのもとで、トップダウンにリダクションするものである。リダクションはそれぞれのゴールについて並列に行なわれる。

プロセス $P_1, \ldots, P_n$ の並列実行は以下では次のように表記する。

$$\mathbf{parbegin} P_1 // P_2 // \ldots // P_n \mathbf{parend}$$

操作的意味論の手続きが扱うデータ構造について: 本稿で与える操作的意味論は、入出力履歴や後に定義する単一化ネット等、半整列順序構造をデータ構造として扱う。さらに後に述べるようにここでは、半整列構造をプロセス間で共有し処理する機能を用いる。 ゆえに、操作的意味論の記述のためには、半整列構造のようなデータ構造を記述できる機能が必要となる。

各プロセスの通信について: ここで与える操作的意味論は、先に述べたように GHC プログラムにおける無限プロセスの動作を記述するものである。したがって各ゴールを処理するプロセスは、各ゴールの計算をすべて終了させた後にその結果として無限な半整列集合を出力するわけにはゆかず、実行中に次々と他のゴールを処理しているプロセスから出力される部分的な結果を読みとり、それを用いて各ゴールにおける計算を進める。このように、各ゴールを処理するプロセスを記述するためには、半整列集合の元を実行中に生成しながら他のプロセスに送るようなプロセス間通信を記述できる機能が必要となる。通常の並列プログラミングでは、このようなプロセス間通信機能を記述するものとして、ストリーム通信の機能が多くの場合用いられる。ストリーム通信の機能を記述できる言語としては、Kahn Network [Kahn 77], Stella [Kuse 86], A'UM, [Yoshida 88] 等が提案されている。また GHC 等の並列論理型言語では、リストを用いてストリーム通信を記述することが通常行なわれる。これらの言語で提案されているストリームは、各データの書き込まれたタイミングによって線形に順序付けられた並びである。この書き込みのタイミングによる順序を用いて読み出しの制御を行ない、プロセス間の同期をとっている。

ここでは、プロセス間の通信で送られる要素は半整列集合の元に限られるものと考える。そこでここではプロセス間通信を、書き込みのタイミングによって順序を用いて制御する構造ではなく、要素が属する領域の上にもともと定義されていた順序を用いて書き込み / 読み出しを制御する構造を用いて行なう。言い替えれば、ここで言う通信用変数とは集合型の変数であり、なおかつその集合の上に定義されている順序によって制御される書き込み / 読み出しの操作が定義されているものと考えることができる。

すなわち通信用変数 S のプロデューサとなるプロセスは、要素を次々と生成し S に書き込む。このとき S に既に書き込まれた元より小さくない元だけを書き込むことができる。コンシューマ側では S の元を読み込む際、読み残している元のうち極小なものを読みこむことができる。

ここで用いられる通信用変数はプロデューサは 1 つのプロセスに限られるが、コンシューマは複数のプロセスに分かれる。プロデューサーとなるプロセスでは出力変数として宣言される。

出力変数に対する書き込みの操作を次のように表記する。

$$\mathbf{put}(A, S)$$

新たな要素 A を出力変数 S に加える動作である。この動作は A が S に既に書きこまれた要素よりも小さくないとき、常に実行できる。

一方、入力変数に対する操作は次のように表記する。

$$\mathrm{get}(S, X)$$

Sに自分がまだ読み込んでいない要素があれば、そのなかで極小なものを選んで内部変数Xに読み込む操作である。Sに目新しい要素がない場合はサスペンドする。ここでは要素の選択にはfairnessを仮定している。すなわち、書き込まれた要素はコンシューマが何らかの理由で停止しない限り、いつかは読まれる。この仮定はGHCのインフォーマルなセマンティクスにおいて、出力された束縛は、いつかは他のプロセスに届くという仮定に対応している。

ここで、あるプロセスの出力変数が2つ以上のプロセスから入力変数として参照されていたとする。この場合ひとつのプロセスがある要素を読み込んでも、スタックやバッファのようにその要素が取り除かれるわけではない。したがって、もう一方のコンシューマプロセスがアクセスしている状態には何の影響も与えないものとする。すなわち複数のコンシューマがひとつの変数を共有して読む場合、各プロセスにそのコピーが配られる、あるいはすべてのコンシューマが自分に固有の入力ヘッドを持つものであると考えられる。

プロセスの動的生成 / 消滅:

以下で述べる操作的意味論では、各ゴールに対応して、それをリダクションするプロセスが起動される。あるゴールについてのリダクションが進むと、サブゴールが呼び出され、それぞれのサブゴールについてリダクションを続けるために再び各サブゴールに対応して新たにプロセスが起動される。このように、並列導出の手続きの記述のためには、プロセスの動的な生成 / 消滅を記述することが可能な言語が必要となる。 ここでは、プロセスの動的生成 / 消滅はプロセスの並列実行を起動する手続きの再帰呼び出しによって記述する。

3.2 操作的意味論

本節では、操作的意味論の基となる並列トップダウン導出の手続きについて主に述べる。まず準備として、この手続きで使われる入力単一化ネットの概念について説明する。

単一化ネット

GHCの場合、通常のProlog等のようにゴール節に対してはじめからすべての入力代入が与えられて計算が実行されるのではなく、実行中に外部との通信を行なうことによって、入力代入が与えられながら実行が進むようなプログラミングを想定している[Ueda 88b]。 実行中に環境から与えられる入力代入とそのタイミングを記述するのが入力単一化ネットである。(以下単に単一化ネットと呼ぶ。) [Ueda 88b] では今着目しているゴール節について、外部環境とのやりとりを、環境をゴール節を観測しているひとつのプロセスと考え、この観測者による入出力の記録が入出力代入(の対)の系列によって記述し、これをトランザクションと呼んでいた。

本稿で扱う単一化ネットはこれとは以下の点で異なる。すなわち、[Ueda 88b] では入出力を環境の立場から観測したものであったのに対し、ここでは着目しているゴール節の立場から観測したものとなっている。すなわち、実行中のある時点で走っているゴールそれぞれが観測者であり、したがってここでは単一化ネットは生起した順序に線形に順序つけることはしない。かわってここでは、各ゴールの起動時からの観測結果をまとめた半整列構造を用いる。

ここでは単一化ネットの各要素である観測事実は、そのゴール節の外側の環境である観測時点までに実行された単一化ゴールの集りとして表現される。このように表現することによって、各単一化ネットの要素間の順序は、ガード付ストリームにおける順序と同様に、集合の包含関係として自然に定義される。

形式的には次のように定義される。

定義 11. (単一化ネット)

代入式の有限集合の集合$T = \{\sigma_1, \sigma_2, \sigma_3, \ldots, \}$が、次の条件を充たすとき単一化ネットと呼ぶ。

1) Tの任意の下に閉じた有限部分集合T'について、$\bigcup_{\sigma \in T'} \sigma$は代入を定義する。このように定義された代入を$T$のプレフィックスと呼ぶ。

2) 任意のσ_iについて、$U \in \sigma_i$かついかなるσ_iより少さいσ_jについても$\sigma_j \not\models U$となるUが少なくともひとつ存在する。

ここで、あるひとつの実行を考えたとき、単一化ネットは着目しているゴール又はゴール節に対して、それ自身の立場から観測結果をまとめた固有なものがそれぞれ考えられる。したがって、以下に述べる手続きでは、実行中に出現するゴール又はゴール節それぞれに対して単一化ネットはそれぞれ管理される。例えば、次のようなゴール節について:

G_1 : `p(X, Y), q(Y, Z)`

このようなゴール節が次のようなプログラムのもとで実行される場合を考える。

D_1 :

```
p(X, Y, W) :- X = 1 | Y = 1, W = 1
q(Y, Z) :- Y = 1 | Z = 1
```

このとき、`X = 1`が外部から入力された実行を考える。このときG_1全体の立場での、`X`が具体化された時点での観測結果は{`X = 1`}となる。一方、`q(Y, Z)`というゴールの立場からは、G_1の環境に加えて、`p(X, Y)`から与えられる束縛情報も外部からの入力として観測される。したがって`q(Y, Z)`の実行が終わった時点での観測結果は、{`X = 1, Y = 1, W = 1`}となる。ここで、`q(Y, Z)`に出現しない`W`についての具体化が観測結果に含まれていることに注意されたい。すなわちあ

るゴールから見たとき、環境で実行された単一化ゴールすべてが観測結果に含まれ、必ずしも引数に出現する変数に対する具体化だけが観測されるのではない。

一方、今着目しているゴール節のサブゴールの間だけで共有されるような内部のローカルな変数については観測結果には含まれない。例えば次のような節があったとき、

```
r(X, Y) :- X = f(X1) | s(X1, Z), t(Z, Y)
```

このとき r(X, Y) の単一化ネットには、X を具体化する要素より大きい要素として X1 を具体化するものが考えられる。しかしながら Z を具体化するようなものは、いかなる単一化ネットの要素としても考えることはできない。

今着目しているゴールの単一化ネットの最小元とは、そのゴールが起動された時に与えられた入力代入 (をつくりだす単一化ゴールの集合) であると考えられる。

導出手続き

トップレベルの手続きは *RUN_GOAL_CLAUSE* で、ゴール節 $G_1,\ldots,G_n$ をプログラム D、単一化ネット T_0 のもとで実行したときのトレース GU_0 をつくるものである。

procedure:
RUN_GOAL_CLAUSE
(入力:$[G_1,\ldots G_n]$, 出力:D, T_0, GU_0) :
begin
parbegin
RUN_GOAL(G_1, D, T_1, GU_1)//
MAKE_TRANSACTION
$([GU_2,\ldots,GU_n], T_0, T_1)$//
RUN_GOAL(G_2, D, T_2, GU_2)//
MAKE_TRANSACTION
$([GU_1, GU_3,\ldots,GU_n], T_0, T_2)$//

$\ldots$

RUN_GOAL(G_i, D, T_i, GU_i)//
MAKE_TRANSACTION
$([GU_1,\ldots,GU_{i-1}, GU_{i+1}, GU_n], T_0, T_i)$//

$\ldots$

RUN_GOAL$(G_n, D.T_n, GU_n)$//
MAKE_TRANSACTION
$([GU_1,\ldots,GU_{n-1}], T_0, T_n))$//
MERGE_TRACE$([GU_1,\ldots,GU_n], GU_0)$
parend
end

ここで *MAKE_TRANSACTION* は、各ゴールの実行環境を表わすトランザクションをつくりだす手続きである。すなわち先の例にも述べたように、各 G_i の実行環境 T_i は T_0 によってゴール節の外側から与えられるトランザクション T_0 に加えて、G_i と並列に走る各ゴール $G_1,\ldots G_{i-1}, G_{i+1},\ldots,G_n$ が実行中につくりだす束縛をも参照する。並列に走るゴールが出力する束縛は、それぞれのゴールを走らせるプロセス *RUN_GOAL* がつくりだすガード付単一化の集合の対 $[GU_1,\ldots,GU_{i-1}, GU_{i+1}, GU_n]$ を参照することによって得られる。

MAKE_TRANSACTION はガード付単一化の集合から単一化ネットに変換する手続きで次のように定義される。

procedure:
MAKE_TRANSACTION
(入力:$[GU_1,\ldots,GU_{i-1}, GU_{i+1}, GU_n], T_0$ 出力:T_i)
parbegin
repeat
  get(X, T_0);
  put(X, T_i)
end_repeat //
repeat
  get(GU_1, gu);
  $U := gu$ のボディ部分;
  put(U, T_i)
end_repeat//

$\ldots$

repeat
  get(GU_{i-1}, gu);
  $U := gu$ のボディ部分;
  put(U, T_i)
end_repeat//
repeat
  get(GU_{i+1}, gu);
  $U := gu$ のボディ部分;
  put(U, T_i)
  end_repeat//

$\ldots$

repeat
  get(GU_n, gu);
  $U := gu$ のボディ部分;
  put(U, T_i)
end_repeat
parend
end

MERGE_TRACE は各ゴールの実行結果 $GU_1,\ldots,GU_n$ をマージして、GU_0 に出力する。

procedure:
MERGE_TRACE
(入力: $GU_1,\ldots,GU_n$出力: GU_0)

parbegin
repeat
　get(GU_1, X_1);
　put(X_1, GU_0)
end_repeat//

...

//
repeat
　get(GU_n, X_n);
　put(X_n, GU_0)
end_repeat
parend
end

RUN_GOAL は中心となる手続きで、ゴール G をプログラム D、単一化ネット T のもとで実行したときのトレースのボディ部分 GU を出力する。以下代入式 $U : X = \tau$ とするとき、テスト式 $X ?= \tau$ を $U?$ で表記する。

procedure:
RUN_GOAL
(入力:G, D, T出力:GU)

$S_0 := \phi$;
repeat
　get(T, σ);
　$S_0 := \sigma \cup S_0$;
until
　begin
　　D から次のような条件を充たす節のある renaming:

C : $H :- U_{g1}, \ldots, U_{gm} | U_{b1}, \ldots, U_{bh}, B_1, B_2, \ldots, B_n$

　　が選べる。
　　(1) $G = \sigma_0 H$ という形の代入が存在する。
　　(2) すべての U_{gi} について $\sigma_0 \cup S_0 \models U_{gi}$。
　end;
parbegin
if $S_0 \cup |GU| \models U_{b1}$ **then**
　put($< \{U_{g1}, \ldots, U_{gm}\} | U_{b1} ? >, GU$)
　else if U_{b1} は $S_0 \cup |GU|$ と無矛盾
　　then
　　put($< \{U_{g1}, \ldots, U_{gm}\} | U_{b1} >, GU$)
　　else *fail*//

...

//
if $S_0 \cup |GU| \models U_{bh}$ **then**
　　put($< \{U_{g1}, \ldots, U_{gm}\} | U_{bh} ? >, GU$)
　else if U_{b1} は $S_0 \cup |GU|$ と無矛盾
　　then
　　put($< \{U_{g1}, \ldots, U_{gm}\} | U_{bh} >, GU$)
　　else *fail*//
repeat
　get(T, σ);
　put(σ, T_{body})
end_repeat//
RUN_GOAL_CLAUSE
　([($\{U_{g1}, \ldots, U_{gm}\} \cup S_0$)$B_1, \ldots,$
　($\{U_{g1}, \ldots, U_{gm}\} \cup S_0$)$B_n$], D, T_{body}, GU_{body})//
UPDATE_STREAM(GU_{body}, C, GU)
parend
end

上の手続きで最初の **repeat - until** 文は、単一化ネットから入力を次々と読みとり、ゴールがコミットできる節 C を捜すステップである。次の **parbegin** 文はボディ部分の実行である。**parbegin** の最初の h 個のプロセスは、C のボディ部分に出現する単一化ゴール $U_{b1}, \ldots, U_{bh}$ についての処理である。各 U_{bj} について、既にその実行環境で観測された束縛から含意されれば、U_{bj} はテスト式として実行されたものとして GU につけ加えられる。また U_{bj} が今迄に観測された束縛と無矛盾でかつ含意されないならば、新たな束縛を出力する単一化が実行されたものとして GU につけ加えられる。U_{bj} が今迄の観測結果と矛盾すれば実行は失敗する。

次の **repeat** 文はボディ部分のサブゴール $B_1, \ldots, B_n$ を実行するための環境となる単一化ネットをつくるプロセスである。

次の *RUN_GOAL_CLAUSE* は、$B_1, \ldots, B_n$ を実際に起動している。最後の *UPDATE_STREAM* は *RUN_GOAL_CLAUSE* によって出力されてくる $B_1, \ldots, B_n$ の実行結果を受けとり、GU を更新するプロセスで次のように定義される。

procedure:
UPDATE_STREAM
(入力:GU_{body}出力:GU)

repeat
　get($GU_{body}, < \sigma | U_b >$)
　$\sigma' := \sigma \cup \{U_{g1}, \ldots, U_{gm}\} \setminus \{U_{b1}, \ldots, U_{bh}\}$;
　$gu :=< \sigma' | U_b >$;
　put(gu, GU)
end_repeat
end

4　宣言的 / 不動点的意味論との関係

本節では前節で述べた操作的意味論と、[Murakami 88] で導入した宣言的 / 不動点的意味論との関係について、考察する。まず、[Murakami 88] で述べた宣言的 / 不動

点的意味論について単一化ネットを用いて再整理し、その概要を述べる。

4.1 宣言的 / 不動点的意味論の概要

定義 12. (解釈)
$I/O - hist$ の任意の部分集合を解釈と呼ぶ.

定義 13. (トレース)
t を入出力履歴、g をゴールとするとき、t が g のある単一化ネット T によるトレースであるとは、次の (1) (3) が成り立つことをいう。

(1) t は $H : -GU$ という形をしており、単一化ネット T の最小値を σ_0 で、$\sigma_0 H = g$ となる。

(2) 任意の $< \theta | U > \in GU$ について、T のプレフィックス σ が存在し、$\theta \subset \sigma$ となる。

(3) 任意の $< \theta | U > \in GU$ について、U が代入式 $X = \tau$ の場合は T のいかなるプレフィックスも X を具体化しない。また U が判定式 $X? = \tau$ の場合は、あるプレフィックス σ が存在し σX は $\sigma\tau$ に等しい。

ここで写像 σ が変数 X を具体化しないとは、$\sigma X = Y (\in Var)$ かつ $\sigma Z = Y$ となる Z が X 以外に存在しないことをいう.

定義 14. (真なゴール)
I を解釈、g をゴールとするとき、g が I で真であるとは、ある単一化ネット T が存在し、g の T によるトレースが I に含まれることをいう.

定義 15. (真なゴール節)
I を解釈、$g_1, \ldots, g_n$ をゴール節とするとき、ゴール節 $g_1, \ldots, g_n$ が I で真であるとは、各 $g_i (1 \leq i \leq n)$ について単一化ネット $T_1, \ldots, T_n$ が存在し、g_i の T_i によるトレース t_i が I に含まれ、$t_1, \ldots, t_n$ のボディ部分 $GU_1, \ldots, GU_n$ の同期付マージ $GU_1 \| \cdots \| GU_n$ が定義できることである. また空 (すなわち $n = 0$ の場合) なゴール節は常に真であるとする.

定義 16. (節集合のモデル)
ガード付節の集合 D について、解釈 I が D のモデルであるとは、I の任意の元 t についてある節:

$$H :- U_{g1}, \ldots, U_{gm} | X_1 = \tau_1, \ldots, X_h = \tau_h, B_1, \ldots, B_k \in D$$

が存在して、t が次のような形をしていることである。

$$\begin{aligned} H :- \{ & < \{U_{g1}, \ldots, U_{gm},\} | uni(X_1, \tau_1) >, \ldots, \\ & < \{U_{g1}, \ldots, U_{gm},\} | uni(X_h, \tau_h) > \} \cup \\ & ((GU_1 \| \ldots \| GU_n) \bowtie \\ & (\{U_{g1}, \ldots, U_{gm}\}, \\ & \{X_1 = \tau_1, \ldots X_h = \tau_h\})) \downarrow Var(H) \end{aligned}$$

ここで T_i は

$$\{U_{g1}, \ldots, U_{gm}\} \cup \{X_1 = \tau_1, \ldots, X_h = \tau_h\}$$

というプレフィックスをもつ単一化ネットであり、かつ

$$\begin{aligned} \forall < \theta | U > \in & (GU_1 \| \ldots \| GU_n) \bowtie \\ & (\{U_{g1}, \ldots, U_{gm}\}, \\ & \{X_1 = \tau_1, \ldots X_h = \tau_h\}) \end{aligned}$$

について

$\forall U \in \theta$ について $\bigcap_i T_i$ のあるプレフィックスから得られる代入 σ について $\sigma \models U$ 。

GU_i は B_i というゴールの T_i によるトレース $(\in I)$ のボディ部、となる。

ある節集合 D の全てのモデルの集合和をとったものはやはり D のモデルであり、これは D の最大のモデルとなることは定義より容易に示せる。節集合 D のセマンティクスとは D の最大モデルによって定義される。また D の最大モデルは、D から定まるある連続関数の最大不動点によって特徴付けられる。[Murakami 88]

4.2 操作的意味論との関係

直観的には最大モデルとは D の上で実行したすべての正常な計算のパスを集めたものと考えることができる。すなわち、D の最大モデルで真となるゴール節とは、D の上で実行した時に、正常に実行を続けることができるようなゴール節であると考えることができる。操作的意味論との整合性については次のようにまとめることができる。ここでは先にも述べたように無限プロセスの動作を記述することが目的である。したがって操作的意味論で、「証明可能なゴール節」とは通常の Horn 論理型プログラムの証明論的意味論のように有限時間後に停止して反駁木が得られることによってではなく、意味論に定められた手続きが障害なく適用できることによって定義される。このように考えると、操作的意味論と宣言的意味論との関係は次のような健全性 / 完全性の概念にまとめることができる。

健全性 — 操作的意味論で実行を続けることができるゴール節は、宣言的意味論 (最大モデル) の上で真である。

完全性 — 宣言的意味論の上で真であるゴール節は、操作的意味論によって実行を続けることができる。

これらの証明のためには、操作的意味論を定義する導出手続きよって生成されるガード付単一化の集合 (とゴールの対) がプログラムの最大モデルと一致することを示すことが必要となる。

5 まとめと今後の課題

ここではGHCの操作的意味論を、GHCの計算規則をガード付単一化の集合を生成する並列導出の手続きとして記述することによって与えた。これによって、従来の逐次的な状態遷移モデルによる記述のように、プロセスのスケジューリングの公平さや、動作のatomicityについて問題にすることなく、GHCの無限プロセスを含むプログラムのセマンティクスを記述することができるようになった。ここで与えた操作的意味論によって生成されるガード付単一化の集合は、その抽象度において、宣言的意味論で用いられるガード付ストリームより詳しいものとなっている。すなわち、ガード付ストリームでは着目しているゴール節の内部でのローカルな通信を表わす要素は隠れているが、操作的意味論では、これらの要素についても含まれている。

今後、このような抽象度の問題を整理し、健全性/完全性の証明を行なうことが課題として残されている。

謝辞: 本研究をすすめるにあたり、有益な議論をして下さった國藤室長、田中研究員はじめ国際研2)-2)研究室の皆様、上田主任研究員はじめICOT第1研究室の皆様に感謝します。本研究は第五世代コンピュータ・プロジェクトの一環として行なわれた。

参考文献

[Corradini 89] A. Corradini and U. Montanari, Towards a process semantics in the logic programming style, Italian- Japanese-Swedish Workshop onm Concurrent Logic Programming and Constraint Logic Programming, Pisa, Italy, 1989

[de Boer] F. de Boer, J.Kok, C. Palamidessi and J.J.M.M.. Rutten, Semantic Models for a version of PARLOG, Proc of Sixth International Conference of Logic Programming, June , 1989

[Falaschi 88] M. Falaschi and G. Levi, Operational and fixpoint semantics of a class of committed-choice logic languages, Dipartimento di Informatica, Università di Pisa, Italy, Techn. Report, January 1988

[Kahn 77] G. Kahn and D.B. MacQueen, Coroutines and network of parallel Processes, IFIP 77 (1977)

[Kok 89] J. Kok, Semantic Models for Parallel Computation in Data Flow, Logic- and Object-Oriented Programming, Ph. D. Thieses of Free University of Amsterdam, 1989

[Kuse 86] 久世, ストリームを扱う言語とその処理系の研究, 筑波大学 工学研究科博士論文, 1986

[Lloyd 84] J. W. Lloyd, Foundations of logic programming, Springer-Verlag, 1984

[Ohyamaguchi 89] 大山口, 那須, 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, 平成元年電気・情報関連学会連合大会, 32-2, 1989

[Murakami 88] M. Murakami, A New Declarative Semantics of Parallel Logic Programs with Perpetual Processes, Proc. of Int. Conf.on Fifth Generation Computer System 1988, 1988

[Saraswat 85] V. A. Saraswat, Partial Correctness Semantics for CP[↓,|,&], Lecture Notes in Comp. Sci., No. 206, 1985

[Saraswat 87] V. A. Saraswat, The Concurrent logic programming CP: definition and operational semantics, Proc. of ACM Symp. on Principles of Programming Languages, 1987

[Shibayama 87] E. Shibayama, A Compositional Semantics of GHC, Proc. of 4th Cof. JSSST, 1987

[Takeuchi 86] A. Takeuchi, Towards a Semantic Model of GHC, Tech. Rep. of IECE, COMP86-59, 1986

[Shibayama 87] E. Shibayama, A Compositional Semantics of GHC, Proc. of 4th Cof. JSSST, 1987

[Ueda 88a] K. Ueda, Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard. Programming of Future Generation Computers, North-Holland, 1988

[Ueda 88b] K. Ueda and K. Furukawa, Transformation Rules for GHC Programs, Proc. of Int. Conf.on Fifth Generation Computer System 1988, 1988

[Ueda 89] K. Ueda and M. Murakami, Formal Semantics of Flat GHC, 平成元年電気・情報関連学会連合大会, 32-3, 1989

[Yoshida 88] K. Yoshida and T. Chikayama, A'UM - A Stream-Based Concurrent Object-Oriented Language - , Proc. of Int. Conf.on Fifth Generation Computer System 1988, 1988