ICOT Technical Report: TR-723

TR-723

変数束縛の伝播の遅延を許すGHCの計算モデル

田中　二郎、菅野　博靖（富士通）

January, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

変数束縛の伝播の遅延を許すGHCの計算モデル

田中二郎　　菅野博靖
(株) 富士通研究所
Email: {jiro, suga}@iias.flab.fujitsu.co.jp

概要

Guarded Horn Clauses (GHC) の計算モデルとして、『制約伝播の遅延モデル』『制約が局所性を持つモデル』の二つを概観したあと、これらのモデルをより詳細化する形で、『変数束縛の遅延と局所性を許すGHCの計算モデル』を提案する。GHCでは、プログラムの実行中に生ずる『変数束縛の伝播の遅延』や『変数束縛の局所性』を許すような仕様となっているが、本方式は、GHCのそのような性質をうまく反映した計算モデルとなっている。

1　はじめに

並列論理型言語 Guarded Horn Clauses (GHC) [12] [13] は Concurrent Prolog (CP) [6] の言語仕様を検討していく経過のうちで、CP に似た新たな言語として提案されたものである。CP と GHC の言語仕様のもっとも主要な違いは、同期の機構を GHC ではガードの中での受動的なユニフィケーションにより実現していることである。すなわち、通常の能動的なユニフィケーションとは異なり、受動的なユニフィケーションではあらたな束縛をガードの外に輸出するようなユニフィケーションはサスペンドする。

また、GHC では並列／分散環境上での実装を考慮して、プログラムの実行中に生ずる変数束縛の伝播の遅延を許すような仕様となっている。すなわち、変数束縛はプログラムの中で均一ではなく局所性 (Locality) を持っており、プログラムの実行中にあるゴールで生ずる変数束縛が他のゴールに伝わるのに時間がかかる。

これを Ask 制約、Tell 制約という考え方で整理したのは Saraswat である [5]。彼は、『変数の束縛』を『制約』の一種としてとらえた。すなわち "$X = Y$" というゴールが実行されれば、X と Y は等しくなるが、それらは "$X = Y$" という制約がストア (Store) へ課されていると考えることができる。制約には、Ask 制約、Tell 制約の二つがあり、ある制約がストアの制約のなかに含まれているかをチェックする受動的な制約が Ask 制約である。また、ある制約をストアの制約のなかに追加する能動的な制約が、Tell 制約である。

GHC の場合は、ガード部の受動的なユニフィケーションで生ずる制約は、実行中のゴールの局所的な変数束縛環境でそれを満たすかどうかを調べられる (Local-Ask)。またボディ部のユニフィケーションで生ずる制約は他のゴールに伝わるのに時間がかかる (Eventual-Tell)。GHC では、このように、何らのグローバルな情報や、変数伝播の即時性を仮定しなくても無事に計算がおこなわれる仕組となっている。

最近、並列論理型言語の意味論に対しても様々な取り組みがなされるようになってきたが [2]、こうした意味論では、並列／分散環境での実装に対応する操作的な意味論は、cc (concurrent constraint) [5] を除けば、ほとんど考察されていない。とくに GHC では、並列／分散環境で実装が容易なように言語設計がなさているが、既存の意味論ではこうした特徴を浮き彫りにするに至っていない。本稿では、そのような性質をうまく反映した計算モデルを提案する。

2 制約伝播の遅延モデル

最初に、GHC の計算モデルを説明するための準備として、Configuration という概念を説明する [1]。
Configuration とは、プログラムの計算状態を表現するための『構造』であり、通常 $< G, C >$ の形式で表現される。ここで、G はゴール列であり、C は制約である。

さて、変数束縛の伝播の遅延の表現であるが、Shapiro は反代入 (Antisubstitution) で特徴づけている [8]。反代入を Configuration の遷移規則として記述すると以下のようになる。

Antisubstitution rule

$$\frac{}{< G_1 \ldots \wedge G_k \wedge \ldots G_n, C > \longrightarrow < G_1 \ldots \wedge G_k[Y/X] \wedge X = Y \wedge \ldots G_n, C >}$$

なおここで、Y は、ゴール列 $G_1 \ldots G_n$ に表れない変数であり、$G_k[Y/X]$ は、G_k の変数 X を Y で置き換えて得られるゴールである。

しかしながら、この反代入は、変数束縛の伝播の遅延モデルとしては強過ぎる定義である。なぜなら、GHC では、変数束縛の伝播の『遅延』はあっても、反代入で特徴づけられるような計算の逆行はないからである。

Saraswat はバッファ付きストアモデルを提案した [5]。ここでは Configuration は、$< G, C_g, C_b >$ の三つ組で表現される。ここで、C_g は、グローバルなストアの制約であり、C_b はバッファにある制約である。このモデルでは、ゴールの計算によって生じた制約は、まずバッファに格納され、バッファからストアに徐々に制約が移動する。これを遷移規則の形で記述すると以下のようになる。

Ask transition rule

$$\frac{\Sigma_0 \models C_g \rightarrow (\exists \bar{x})(C_{G=H} \cup C_C)}{< G \wedge G_{rest}, C_g, C_b > \longrightarrow < G_{rest} \wedge B, C_g, C_b \cup C_{G=H} \cup C_C >}$$

ここで Σ_0 は、与えられた制約系であり、$\bar{x}$ は、ゴール G のリダクションに使われる候補節 $H \leftarrow C \mid B$ に表れる変数列を示している。すなわち、ヘッド・ユニフィケーションとガード部によって作られる制約が、既にシステムの持つ制約の一部であり、新たに制約をつくり出すことがなければ、リダクションが行なわれる。

Tell transition rule

$$\frac{}{< X = Y \wedge G_{rest}, C_g, C_b > \longrightarrow < G_{rest}, C_g, C_b \cup C_{X=Y} >}$$

これは、実行されるゴールがユニフィケーション・ゴールである場合に対応する。この場合、バッファにはユニフィケーションに対応する制約 $C_{X=Y}$ が格納される。

Globalizing rule

$$\frac{C_b = C \cup C_{b'} \; and \; \Sigma_0 \models (\exists) C_g \cup C}{< G, C_g, C_b > \longrightarrow << G, C_g \cup C, C_{b'} >}$$

これはバッファにある制約 C_b から制約の一部 C がグローバル環境に移動することに対応している。

Failure invoking rule

$$\frac{C_b = C \cup C_{b'} \; and \; \Sigma_0 \models \neg(\exists) C_g \cup C}{< G, C_g, C_b > \longrightarrow fail}$$

これはバッファにある制約がグローバル環境に移動しようとしたが、グローバル環境 C_g と移動する制約 C を合わせたものが可解でないので計算が失敗する場合に対応する。

しかしながら、このバッファ付きストアモデルでは、変数束縛の遅延は表せても、変数束縛がプログラムの中で均一ではないいわゆる『局所性』は表現することができない。

3 制約が局所性を持つモデル

ストアに蓄えられる制約が局所性を持つ場合について、Saraswat は分散ストアモデルを提案している [4]。ここでは、局所性の単位としてゴールを考えている。

このモデルでは Configuration は、$< T, C_b >$ の形で表現される。ここで、T はローカル環境の AND 結合であり、C_b はバッファにある制約である。また、個々のローカル環境は $< G, C_l >$ の形で表現される。G はゴール列であり、C_l はそのゴール列が知っている『局所的な』制約である。

この場合、遷移規則を次のように定義できる。

Ask transition rule

$$\frac{\Sigma_0 \models C_l \rightarrow (\exists \bar{x})(C_{G=H} \cup C_C)}{<< G, C_l > \wedge T, C_b > \longrightarrow < T \wedge \bar{B}, C_b \cup C_{G=H} \cup C_C >}$$

なお、ここで $\bar{B}$ は、ゴール G のリダクションに使われる候補節 $H \leftarrow C \mid B$ で B が $B_1 \ldots B_n$ であるとすると、それぞれの B_j $(1 \leq j \leq n)$ についてローカル環境 $< B_j, \phi >$ を作ったものの列である。

Tell transition rule

$$\frac{}{<< X = Y, C_l > \wedge T, C_b > \longrightarrow < T, C_b \cup C_{X=Y} >}$$

実行されるゴールがユニフィケーションである場合、バッファにはユニフィケーション・ゴールに対応する制約が格納される。

Constraint transmission rule

$$\frac{\Sigma_0 \models C_b \rightarrow C \; and \; \Sigma_0 \models (\exists) C_l \cup C}{<< G, C_l > \wedge T, C_b > \longrightarrow < T \wedge < G, C_l \wedge C >, C_b >}$$

これはバッファにある制約 C_b から制約の一部 C がローカル環境に複写されることに対応している。

Failure invoking rule

$$\frac{\Sigma_0 \models C_b \rightarrow C \; and \; \Sigma_0 \models \neg(\exists) C_l \cup C}{<< G, C_l > \wedge T, C_b > \longrightarrow fail}$$

これは制約がローカル環境に移動する場合で、すでにあるローカル環境 C_l と移動する制約 C を合わせたものが可解でないので計算が失敗する場合に対応する。

この、局所性のモデル化については、一つ一つのゴールがローカルな制約を持つ形となっている。また、ローカルな制約は空から始まり、徐々に共通のバッファから制約が複写される。

4 制約伝播の遅延と局所性を許す計算モデル

そこで、こうした従来のモデルをより詳細化、現実化した制約伝播の遅延と局所性を許す計算モデルを提案する。

ここでは、局所性を持つ計算モデルをより現実的なものとするために、あるゴール・グループが共同でローカル環境を共有すると仮定する (これは、GHC の分散インプリメントに対応する)。ローカル環境を作るためには、そのゴールの後にプラグマ @ を陽に付けて指定すると仮定する [7]。例えば、定義節

$$H \leftarrow C \mid B_1 \; B_2 @ \; B_3$$

の場合であれば、あるゴールがこの定義節を用いてリダクションされるとき、B_2 については新しくローカル環境が作られるが、B_1、B_3 については、いままでの古いローカル環境の中で処理が進む。

このモデルでは『変数』や『制約』に局所性を導入する。すなわち、GHC の計算の過程で、『変数』や『制約』が作られていくが、計算の過程で作られる『変数』は、作られたローカル環境に『ホーム・ポジション』を持つと考える。また、計算の過程で作られる『制約』についても、どこか一つのローカル環境の中にしまわれると考える。

このモデルでは Configuration は、$< T, M_b >$ の形で表現される。ここで、T はローカル環境の AND 結合であり、M_b はグローバルなメッセージ・バッファである。グローバルなメッセージ・バッファは、宛名を持つメッセージをたくわえており、やがて宛名のローカル環境に配達される。また、個々のローカル環境は $< Id, G, C_l, C_{ext} >$ の形で表現される。Id はローカル環境の識別子、G はゴール列、C_l はローカル環境の格納している制約、C_{ext} は外部変数の制約である。ここで、C_{ext} はローカル環境が輸入している外部変数の値を格納する機能を持っており、例えば、計算の過程で、外部変数 V の値が t であることがわかったとき、C_{ext} には $V = t$ という制約が新たに書き加えられる。

このモデルでは、当然ながら遷移規則は従来のモデルよりやや複雑になる。まず **Ask transition rule** であるが、これは、以下の 3 つの規則になる。

Ask transition rule (1)

$$\frac{\Sigma_0 \models C_l \cup C_{ext} \rightarrow (\exists \bar{x})(C_{G=H} \cup C_C)}{<< Id, G \wedge G_{rest}, C_l, C_{ext} > \wedge T, M_b > \longrightarrow \\ < T \wedge < Id, G_{rest} \wedge B, C_l \cup C_{G=H} \cup C_C, C_{ext} >, M_b >}$$

この遷移規則は、あるローカル環境内でゴールがリダクションされることに対応している。すなわち、ヘッド・ユニフィケーションとガード部によって作られる制約が、ローカル環境と変数テーブルの持つ制約の一部であり、新たに制約をつくり出すことがなければ、ゴール G は、候補節
$H \leftarrow C \mid B$ によりリダクションされ、B が新たに、ローカル環境のゴール列に加えられる。

Ask transition rule (2)

$$\frac{\Sigma_0 \models C_l \cup C_{ext} \rightarrow (\exists \bar{x})(C_{G=H} \cup C_C)}{<< Id, G@ \wedge G_{rest}, C_l, C_{ext} > \wedge T, M_b > \longrightarrow \\ < T \wedge < Id, G_{rest}, C_l, C_{ext} > \wedge < Id_{new}, B, C_{G=H} \cup C_C, \phi >, M_b >}$$

これは、ゴールがリダクションされるときに新たにローカル環境が生成される場合である。すなわち、新たに生成されたローカル環境には、新しい識別子が割り当てられ、ゴール列には、リダクションされたゴール B が、ローカルな制約にはヘッド・ユニフィケーションとガードの制約が、生成される。

Ask transition rule (3)

$$\frac{\neg(\Sigma_0 \models C_l \cup C_{ext} \rightarrow (\exists i)(\exists \bar{x}_i)(C_{G=H_i} \cup C_{C_i}))}{<< Id, G \wedge G_{rest}, C_l, C_{ext} > \wedge T, M_b > \longrightarrow \\ < T \wedge < Id, G \wedge G_{rest}, C_l, C_{ext} >, M_b \wedge M_G >}$$

これは、与えられた制約 $C_l \cup C_{ext}$ では、リダクションできる候補節 $H_i \leftarrow C_i \mid B_i$ が存在しない、すなわち、情報が不十分でゴール G がリダクションできない場合である。このときには、外部変数で値がわからないものについて、最新の情報を集めるため、他の環境に外部変数の値を聞く必要がある。

これをおこなうのが、M_G で、get_value メッセージの列からなっている。get_value メッセージは、$< Id_V, get\_value(V, Id_R) >$ の形をしており、Id_V はこのメッセージの送り先（すなわち、変数 V のホーム・ポジションのローカル環境）、V は値の欲しい変数、Id_R は変数 V の結果を返すべき（すなわち、現在の自分の）ローカル環境を示している。この get_value メッセージを受けとった先では、$reply$ メッセージを Id_R に返すが、それもメッセージ $< Id_R, reply(V, Value) >$ の形式で行なわれる（このメッセージを受けとったローカル環境の C_{ext} には、制約 $V = Value$ が加えられる）。

なお、この遷移規則については、G が $G@$ の形式のときも同様である。

つぎに、**Tell transition rule** であるが、これは以下の 2 つの遷移規則になる。

Tell transition rule (1)

$$\frac{\Sigma_0 \models C_l \cup C_{ext} \cup C_{X=Y}}{<< Id, X = Y \wedge G_{rest}, C_l, C_{ext} > \wedge T, M_b > \longrightarrow \\ < T \wedge < Id, G_{rest}, C_l \cup C_{X=Y}, C_{ext'} >, M_b \wedge M_{X=Y} >}$$

実行されるゴールがユニフィケーション・ゴールである場合、まず、制約 $C_{X=Y}$ とメッセージ $M_{X=Y}$ を求める。$C_{X=Y}$ や $M_{X=Y}$ がどのようなものになるかは、X や Y がローカルな変数であるか、外部変数であるか、あるいはすでに値が求まっているかなどにより異なる。すなわち、ユニフィケーションがローカルに処理できる場合には、そのユニフィケーションに対応する制約が $C_{X=Y}$ であり、メッセージ $M_{X=Y}$ は空になる。また、ローカルに処理できない場合には、逆に $C_{X=Y}$ は空になり、メッセージ $M_{X=Y}$ には $< Id, unify(X,Y)) >$ もしくは、$< Id, unify(X, func) >$ などの形式のメッセージが入る。ここでは、$C_{X=Y}$ をローカルな制約系に加えることが矛盾しないかをチェックし、制約を加えることが可能であれば、ローカルな制約 C_l に加えられる。

Tell transition rule (2)

$$\frac{\Sigma_0 \models \neg(C_l \cup C_{ext} \cup C_{X=Y})}{<< Id, X = Y \wedge G_{rest}, C_l, C_{ext} > \wedge T, M_b > \\ \longrightarrow fail}$$

これは、ユニフィケーション・ゴールを実行する場合、ユニフィケーション・ゴールに対応する制約をローカルな制約に加えたものが可解でないので計算が失敗する場合に対応する。

残る問題は、メッセージの処理である、まず各ローカル環境からのメッセージは、グローバルなメッセージ・バッファに送られるが、それを宛名のローカル環境に送ってやる必要がある。これを行なうのが以下の遷移規則である。

Message transmisson rule

$$<< Id, G, C_l, C_{ext} > \wedge T, < Id, Mes > \wedge M_b > \longrightarrow < T \wedge < Id, Mes \wedge G, C_l, C_{ext} >, M_b >$$

すなわち、各ローカル環境は、グローバルなメッセージ・バッファにあるメッセージの宛名が自分であれば、メッセージを自分のゴール列に取り込む。ここでは、簡単化のためメッセージもゴールとして処理できると考えている。しかしながら、この場合、こうしたメッセージに対する処理が組み込み述語として、制約系の中に用意されていることが前提となる。

5 計算モデルの詳細化

以上、制約伝播の遅延と局所性を許す計算モデルについて、その枠組を遷移規則により記述したが、ローカル環境の間で交換されるメッセージは、制約系によりどのように処理されるのかという点に関しては十分には述べなかった。そこで、ここではそういった処理を行なうための計算モデルの詳細化について簡単に述べる。

まず、変数の表現であるが、本方式では計算の過程で作られる変数は、作られたローカル環境に『ホーム・ポジション』を持つ。したがって変数は $< G_{Id}, L_{Id} >$ の二つ組として表現される必要がある。 ここで、G_{Id} は変数の作られたローカル環境を示す識別子であり、L_{Id} はローカル環境の中での識別子である。

このように変数を表現した場合、ある変数がローカルな変数であるか、あるいは外部変数であるかを区別するのは簡単である。すなわち、変数の G_{Id} がローカル環境の Id と等しければその変数はローカルであり、等しくなければその変数は外部変数である。

つぎに、G_{Id} の間や、L_{Id} の間に、何らかの形で全順序関係を定義すれば、任意の二つの変数の間にも全順序関係 "$\prec$" が定義できる。また、変数と関数の間では、" 関数 $\prec$ 変数 " とすれば、ある変数 V について、与えられた制約系 Σ_0 の中での等号による同値類が定義でき、そのなかで最小の元（代表元）$min(V)$ を求めることが可能となる。その場合、$< Id_V, get\_value(V, Id_R) >$ メッセージは、変数 V の代表元を求め、ローカル環境 Id_R に返す命令として定義できる。

また、本方式では、計算の過程で作られる制約について、どこか一つのローカル環境の中にしまわれると仮定したが、例えば、ユニフィケーション "$X = Y$" という制約であれば、X の代表元と Y の代表元のうち、G_{Id} の大きな方のローカル環境に制約 $min(X) = min(Y)$ をしまうと定義することができる。

6 まとめ

GHC の計算モデルとして、『制約伝播の遅延モデル』『制約が局所性を持つモデル』の二つを概観したあと、これらのモデルをより詳細化する形で、『変数束縛の遅延と局所性を許す GHC の計算モデル』を提案した。この計算モデルは、まだ基本的なモデルの段階であるが、[3] [10] で我々が提案したような、GHC の分散計算モデルの基本メカニズムを具備している。

また、本方式の応用であるが、最近、我々は、GHC のメタやリフレクションについて研究 [11] [9] をおこなっており、とくに、分散環境でのリフレクションは、現在の重要な研究テーマの一つである。本稿のモデル化は、この問題についての一つのアプローチを提供すると思われる。

なお、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ものである。

参考文献

[1] M.J. Maher: Logic Semantics for a Class of Committed-choice Programs, Logic Programming: Proceedings of the Fourth International Conference, The MIT Press, Vol.2, pp.858-876, 1988.

[2] 村上昌己: 並列論理型言語の形式的意味論, 情報処理, Vol.32 No.7, pp.819-838, 1991 年 7 月.

[3] 大原有理, 鳥居悟, 小野越夫, 岸下誠, 田中二郎, 宮崎敏彦: FGHC ソフトウェアシミュレータの試作, The Logic Programming Conference '86, pp.93-101, ICOT, Jule 1986.

[4] V.A. Saraswat: A Somewaht Logical Formulation of CLP Synchronisation Primitives, Logic Programming: Proceedings of the Fifth International Conference and Symposium, The MIT Press, Vol.2, pp.1298-1314, 1988.

[5] V.A. Saraswat: Concurrent Constraint Programming Languages, Ph.D. Thesis, Carnegir-Mellon University, January 1989.

[6] E. Shapiro: A Subset of Concurrent Prolog and Its Interpreter, ICOT Technical Report, TR-003, 1983.

[7] E. Shapiro: Systolic Programming: A Paradigm of Parallel Processing, Proceedings of the International Conference on Fifth Generation Computer Systems 1984, pp.458-478, ICOT, November 1984.

[8] E. Shapiro: The Family of Concurrent Logic Programming Language, ACM Computing Surveys, Vol.21, No.3, pp.413-510, ACM, September 1989.

[9] 菅野博靖: 並列制約論理型言語における並行リフレクションについて, 情報処理学会研究報告, 91-PRG-3, Vol.91, No.60, pp.17-25, 1991 年 7 月.

[10] J. Tanaka, T. Miyazaki, K. Taki and T. Chikayama: Distributed Implementation of FGHC: Toward the Realization of Multi-PSI System, ICOT Technical Report, TR-159, 1986.

[11] 田中二郎, 的野文夫: リフレクティブ GHC とその実現, The Logic Programming Conference '90, pp.179-189, ICOT, July 1990.

[12] K. Ueda: Guarded Horn Clauses, ICOT Technical Report, TR-103, 1985.

[13] K. Ueda: Guarded Horn Clauses, Ph.D. Thesis, University of Tokyo, March 1986.