5B-3 データフローマシン向きPROLOG型言語に関する一考察

相田 仁，松方 純，鈴木達郎
田中英彦，元岡 達（東京大学工学部）

1. PROLOGとデータフロー

Kowalskiの提案[1]に基いて、一階述語論理のサブセットであるHorn節をプログラミング言語として用いるPROLOGが各地でインプリメントされ、プログラミングの容易さなどの観点から注目を浴びている。[2]

PROLOGにおいて主要部分をなすprocedure declarationは、一般に次のような形をしている。

$$+A(x,y) - B(x,y) - C(x,y) \quad \cdots\cdots ①$$
$$+A(x,y) - D(x,y) - E(x,y) \quad \cdots\cdots ②$$

上式は$((B \wedge C) \vee (D \wedge E)) \Rightarrow A$を意味する。いま、Bが成り立つようなx,yの範囲と、Cが成り立つx,yの範囲の両方がわかれば、①より、これらの共通部分に対してAが成り立つことがわかる。そこで、各述語が成り立つために引数の間に成り立つべき関係（ここではxとyの関係）を、あるいは、各述語を成り立たせる解の集合を、ひとつのデータと解釈すれば、上記①，②式は図1のようなデータフローグラフに対応づけられる。

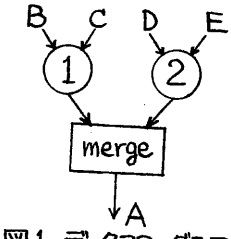

図1 データフローグラフ

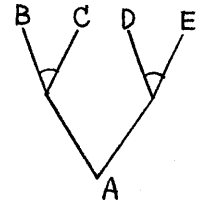

図2 and-or tree

図1において、①のノードにBを成立させる条件とCを成立させる条件が到着すると、それらのandがとられる。②のノードも同様である。mergeノードではこのようにして①，②のノードで得られた条件のorがとられて、最終的にAを成立させるための条件として送り出される。このように、図1のデータフローグラフは、従来から証明過程の解析に用いられているand-or tree（図2）において、引数の相互関係を明確にしたものと考えることもできる。

2. PROLOGの拡張

プログラミング言語としてのPROLOGの最大の問題点は、プログラム実行の停止性が必ずしも保証されない点である。PROLOGにおいて、述語の不成立は、証明過程におけるすべてのchoiceを行ない、そのすべてが失敗することを意味している。しかし述語不成立の場合、一般に証明が無限の深さに落ち込んでゆき、有限時間内に終了しないため、解が存在しないことを確認することができない。同様のことはすべての解を得ようとする場合にも言える。PROLOGでは、通常、解はすべての引数がground termにbindされた状態で得られるため、解集合が無限集合となる場合には、証明の初期の段階で行なわれた選択がback-trackされず、解の一部分しか得られない可能性がある。

そこでデータフローマシンによる並列処理を活かしてこのような問題点を解決するため、次のような2種類のstatementを新たに導入する。

	記法	論理的定義
I.	×P(x,y) ×Q(x,y)	$\forall x \forall y [\sim(P(x,y) \wedge Q(x,y))]$ ……③
II.	∨P(x,y) ∨Q(x,y)	$\forall x \forall y [P(x,y) \vee Q(x,y)]$ ……④

これらの形の文が宣言されているとき、PまたはQのいずれかがgoalとして設定されると、他方も同時にgoalとして設定され、並列に処理を行なう。I.の場合、例えばQが(x,y)のある範囲で成り立つことが先に証明されれば、③よりその範囲でPは成り立たないことがわかるので、Pの証明において探索範囲を縮小することができる。特に、与えられた条件下で常にQが成立することが証明されれば、Pはfailすることがただちにわかり、Pだけを証明する場合には無限の深みに陥る場合でも、有限時間内に証明を終了することができる。

例として述語sumを⑤,⑥により定義する。これに対し、⑦は解の存在しないgoal statementであり、従来のPROLOGでは、⑥の繰り返し適用に無限に深く落ち込んでしまう。これに対して、新たに導入された記法を用いて⑧,⑨のように述語musを宣言しておくと、sumがgoalとして設定されると、同時にmusもgoalとして設定され、並列に処理が開始される。左の枝がx=zなる条件の下でのみ成立することがわかるとこれが右の枝に伝えられる(図3)。x=zの条件下でmusは常に成立。従って⑨よりsumは常に不成立であることが認定され、証明は打ち切られる(図4)。

```
+ sum(x, 0, x)                                  ……⑤
+ sum(x, S(y), S(z)) - sum(x, y, z)             ……⑥   ただし S(x) = x+1
- sum(x, 0, z) - sum(x, S(0), z)                ……⑦        successor function
+ mus(x, S(y), x)                               ……⑧
× sum(x, y, z) × mus(x, y, z)                   ……⑨
```

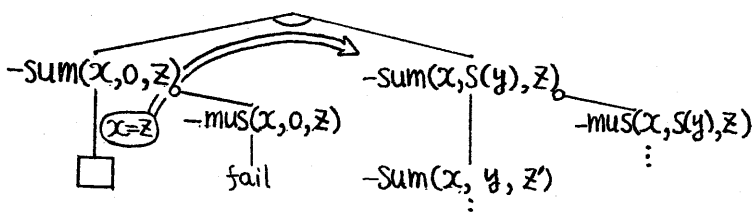

図3 goalの展開

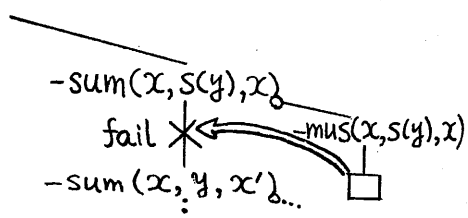

図4 musの成立によるsumの打ち切り

II.についてもI.の場合と成立－不成立の関係が逆になるだけで同様である。

3. インプリメンテーション

前節に述べたような動作をデータフロー的に行なうため、述語の成立条件を担うトークンの他に、不成立条件を担うトークンも考え、解に関する知識が得られ次第、これらのトークンをやりとりしてノード相互間の通信を行なう方針であるが、詳細は別の機会にゆずる。現在、この拡張PROLOGを高レベルデータフローマシン"TOPSTAR"の上に実装すべく準備中である。

1. R.Kowalski : Predicate Logic as Programing Language, Proc, IFIPC, 1974
2. D.McDermott : The PROLOG phenomenon, ACM SIGART, July, 1980
3. 中島秀之 : PARALLEL PROLOG, 第21回 プログラミングシンポジウム, 1980
4. K.Clark & F.McCabe : IC-PROLOG - Language Features, Logic Programming Workshop, 1980