パーソナル逐次型推論マシンψ

― その言語システム ―

近山　隆　　高木　茂行　　坂井　公
（財団法人　新世代コンピュータ技術開発機構）

1. はじめに

ＩＣＯＴで開発中の逐次型推論マシンψは基本的にはPROLOGと同様の機能を有する言語ＫＬ０を機械語として直接実行する高級言語マシンである．オペレーティング・システムを含め，ψ上のすべてのプログラムは最終的にはこのＫＬ０によって実行されることになる．

ＫＬ０の基本実行メカニズムはPROLOGと同様ユニフィケイションとバックトラッキングによるAND-OR木の探索であるが，この他に従来型の計算機と同様のメモリ書込・参照の機能も備えている．この機能は論理型プログラムの立場から見れば「副作用」であり「望ましくないもの」である．しかし，時間の概念を取入れた論理体系に対する効率のよい処理方式が見あたらない現在，実際に時間と共に変化する状態をもつ外界とのインタフェースを行なうプログラムを記述する上では本質的に必要不可欠な機能である．とはいえ，副作用をもつ機能を不用意に用いると，論理型プログラミングの利点を大きく損うことになりやすい．

ＥＳＰはψ上のオペレーティング・システムや種々のアプリケーション・プログラムの記述に用いることを目的とする言語であり，ＫＬ０にコンパイルされて実行される．ＥＳＰは，論理型言語としての整合性をできる限り損わない形で状態の概念を導入するため，状態をもった「対象」という概念を導入し，対象指向機能を取入れた言語になっている．

2. 対象と状態

ＥＳＰでは対象はひとつの公理系であると考える．ある対象のもつ状態に対する問合せは，その公理系に基づいた証明を行なうことに相当する．対象のもつ状態を変更することは対象の表現する公理系を変更することであり，これまでのPROLOGに見られるassertやretract の機能にあたる．もちろん対象は複数個扱うことができるので，いわゆる多世界の機能を持つことになる．述語の呼出しにあたってどの公理系を用いるかは，第一引数として渡される対象によって実行時に決定される．

対象は論理型言語の世界においては個別の定数として扱われ，異なるふたつの対象はたとえ同じ公理系を表わしていてもユニファイできない．ある時点でふたつの対象が表現する公理系が同一であったとしても，それらは将来変更される可能性がある．したがって表現する公理系が同一であることを条件にユニファイすると，いったんユニファイできたもの同志がユニフィケーションをバックトラックしないにもかかわらずユニファイできなくなるという事態が生ずる．これは論理型言語の意味の根幹にかかわることであり，処理に際してもPROLOGの論理型言語としての性格からくる実現手段の選択の自由度を狭め，可能な最適化の範囲を非常に限定することになってしまう．

対象の表現する公理系に対してassertやretract のような任意の変更を許すことは，プログラムの読みやすさを損いやすい．また，処理系の実現上も実行中のプログラム部分の書替についての対処が必要になり，最適化が困難である．このため，従来の処理系では変更は単純実行コードに限定し，コンパイル済コードに関しては禁止している場合が多い．

従来のPROLOG処理系を用いてのプログラミングにおいては状態変化の概念は特に必要としないのが普通である．これは状態の概念が必要不可欠である部分，たとえば入出力にかかわる部分などに対しては特別の組込述語を用意し，PROLOGのプログラムから隠してしまうことによって，状態の概念が表面にでてくるのを避けているためである．したがってassertやretract を用いる局面はかなり限られており，その部分が非効率的な解釈実行によって実行されても全体に及ぼす損失は大きくない．

これに比して，ＥＳＰの場合は計算機システムのもっとも基礎的な部分の記述にも用いる言語である．たとえば，入出力機器の制御など状態の変化が本質的であるプログラムの記述に用いることを考えると，状態概念の記述に正面から取組まざるを得ない．また，実現手段の効率性にもかなり配慮する必要がある．

ＥＳＰでは公理系の変更を「スロット値」と呼ばれる値の変更に限定した．スロット値は特定の述語を通して参照される．スロット値の変更はこの述語に関する公理の変更と考えられる．このようにすると変更を受け得る公理は限定されるので，プログラムの読解性を向上できる．また，スロット値の参照や変更は実際上は記憶語の参照・格納になり，相互排他処理を特に行わずとも実行中のプログラムの書替えは起こらないので，コンパイルも容易で効率的な処理を行える利点もある．

3. クラスと継承

対象のひとつひとつの存在意義はそれぞれが異なる公理系を表現することにある．しかしながら，良く似た公理系を表わす一群の対象がある場合，このような対象群に対して個別に公理系を定義することは，プログラムのモジュラリティを損うことになる．

ＥＳＰでは，スロットの値以外はまったく同一の公理系を表わす対象群に対して，共通する公理群をクラスとして定義できるようにした．個々の対象はいずれかのクラスに属することになる．

スロット値以外の公理群についても，複数のクラス間に共通する部分が多いこともある．たとえばゾウに関する公理群もクジラに関する公理群も同じ哺乳類に関する公理群を含む．ＥＳＰではこのような場合，哺乳類一般に適用できる公理群を「哺乳類」というクラスにまとめ，クラス「ゾウ」やクラス「クジラ」がこの「哺乳類」クラスを継承する，という記述ができるようにした．ＥＳＰでは継承するクラスがひとつだけの単一継承に限定せず，いくつもの親クラスを継承する多重継承ができるようになっている．たとえば「クジラ」は「哺乳類」のほかに「海棲動物」というクラスを同時に継承することができる．

原則として親のクラスのもつ公理は子のクラスの対象にも引き継がれる．論理型言語としては親のもつクローズが子のもつクローズ群に追加される形になる．これによってIS-A階層構造をもつ意味ネットワークが形成されるわけである．

4. デモン

複雑な処理を必要とするプログラムを記述する際には，デモンの考え方は大変有用である．たとえば，マルチ・ウィンドウ・システムの記述ではウィンドウ全体に対する操作（たとえばウィンドウ名の表示）についても何らかの操作が必要になる．ウィンドウを記述する際に付属物に対する操作もすべて記述すれば良いが，それでは付属物の組合せのすべてについて個別にウィンドウに対する操作を記述する必要があり，プログラムのモジュラリティを損う．デモンの考え方を用いれば，ウィンドウに対する操作に対応して付属物の方で必要な操作については付属物の側に記述すればよく，記述量が組合せ的に増加することはない．

ＥＳＰでは，ある述語名と引数個数をもつ公理について，それが適用される直前または直後に実行すべきコードをデモンとして指定することができる．通常の公理の継承が公理の追加，証明探索としてはＯＲであるのに対し，デモンの継承はＡＮＤで行われる．ユニフィケーション結果は前置デモン→公理本体→後置デモンの順に渡されるので，どのような引数で呼ばれたか，結果はどうなったか，をデモンで調べて対応することが可能になっている．

```
% Lock
class lock has
instance
    attribute state := unlocked;
    locked(Lock) :- Lock!state = locked;
    unlocked(Lock) :- Lock!state = unlocked;
    ...
end.

% With a Lock -- MIXIN
class with_a_lock has
instance
    attribute lock is lock;
    before open(Obj) :- :unlocked(Obj!lock);
end.

% Simple Door
class door has
instance
    component state := closed;
    open(Door) :- Door!state := open;
    close(Door) :- Door!state := closed;
    make_way(Door) :- Door!state = open. !;
    make_way(Door) :- :open(Door);
    ...
end.

% Door with a Lock
class door_with_a_lock has
    nature  door, with_a_lock;
end.

% Something with a Door  --  MIXIN
class with_door has
instance
    attribute door is door_with_a_lock;
    make_way(Obj) :- :make_way(Obj!door);
end.

% Simple Room
class room has
    nature container;
instance
    enter(Room) :- :make_way(Room), ... ;
    exit(Room) :- ... ;
end.

% Room with a Door
class room_with_one_door has
    nature room, with_door;
end.
```

図１．ＥＳＰプログラムの例

5. おわりに

ＥＳＰは83年 8月現在クロス・システムが稼働中である．ゆのオペレーティング・システムの開発はこのクロス・システム上で行い，ハードウェア／ファームウェアの完成を待ってセルフ・システム上に移行する予定である．

ＥＳＰの言語仕様の作成は横井室長以下ＩＣＯＴ研究所第三研究室のメンバーの共同作業によるものである．