ＰＩＥにおける集合および多重世界の実現方式

2B-1

相田　仁、◇松原　健二、　田中　英彦、　元岡　達

（東京大学　工学部）

１．はじめに

ＰＩＥは論理型言語をＯＲ並列性にもとづいて高並列に実行する。現在、知識情報処理に有用な各種の機能を、並列性を必要以上に制限することなく、効率よく記述可能な論理型言語に関する検討をすすめている。今回はその中から解の寄せ集めと時間独立多重世界の機能をとりあげて、ＰＩＥにおける機能の実現法を述べる。

２．解の寄せ集め

ＰＩＥにおいては、通常ＯＲ関係にある複数の定義節が並列に適用され、並列して解が求められる。このようにして求められるすべての解を寄せ集めて解の集合を得たい場合がある。一般に解の寄せ集めを行なおうとすると複数の定義節の適用が単なるＯＲではなくなるので何らかの工夫が必要となる。ＰＩＥにおいて各々の解を並列に求めつつ、得られた解を寄せ集めようとすると、次のような３つの方法が考えられる。

①推論木上に解のプールを設ける方法

推論木のノードとして解の寄せ集めを行なうノードを考え、解をsuccess コマンドに乗せて推論木上を送り解をリスト上につないでいくもの。

Conery のＡＮＤ／ＯＲモデルのように推論木の各ノードをプロセスとしてとらえる場合には最も自然な実現法であるが、ＰＩＥにおいてはＡＣがMM内のデータを操作する機能が必要となり、好ましくない。

②部分集合を求める問題に分解していく方法

図１に示すようにゴールｐ（Ｘ）を満たす解の集合を求める場合、ｐ（Ｘ）と単一化可能な定義節が複数存在すれば、それらの各々を満たす解の集合を求め、それらの和集合を求める問題に帰着される。そこで実際にそのようなゴールフレームの書き換えをＵＰにおいて行ない、あとはＡＮＤ並列のテクニックによって処理を進める。具体的には、リテラル間の共有変数である部分解集合に対応したセルを構造メモリ上に用意し、部分解集合を求めて結果をそこにいれる一方、和集合を作るためのリテラルは残りのリテラルと合わせて部分解集合がすべて求まるまで待たせておく。この方式は構造メモリ等に必要とされる機能が少ないが、一旦解の寄せ集めを行なうノードが生成されると、それ以降がすべてＡＮＤ並列を行なうノードとなるのでオーバーヘッドの点で問題がある。

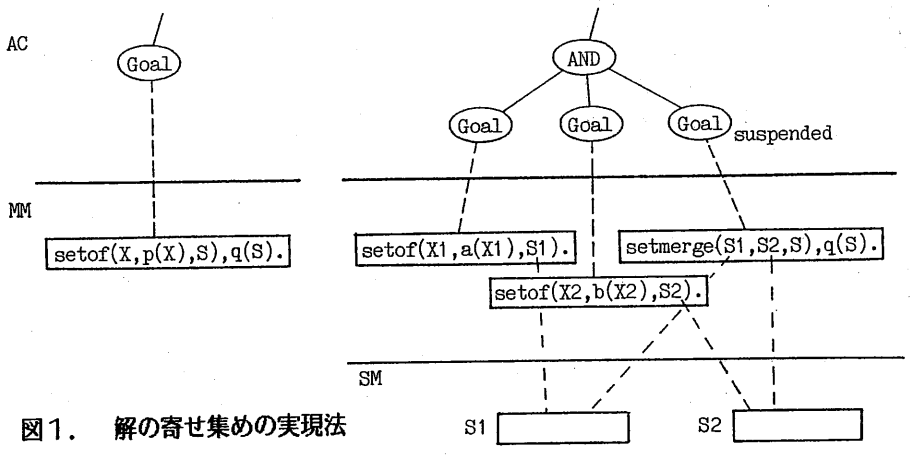

図１．　解の寄せ集めの実現法

③高機能構造メモリを用いる方法

構造メモリに、解の寄せ集め用の高レベル操作を行なう機能を実装すると、より自然な形で解の寄せ集めの実現が可能となる。解の寄せ集めの際には、まず解の寄せ集めを行なうリテラルと他のリテラルを切り分け、解集合に対して構造メモリ内に空集合を作成して識別番号を割り当ててもらう。切り分けたゴールのうち解を寄せ集めるべきゴールについては、解が求まると解集合の識別番号と得られた解を構造メモリに送って蓄積してもらう。

これら３種の方式のうち最終的には③の方式が優れていると考えられるが、構造メモリ機能の実現可能性の検討と並行して②、③の両方式についてシミュレーションを行なっていく予定である。

３．多重世界

述語の定義を動的に変更しつつ問題を解いてゆく多重世界の機能としては、入出力など副作用と関係して時間とともに変わっていく世界と、時間とは関係なく、複数のプロセッサで並列に処理可能な世界の２種類を検討しているが、ここでは後者の時間独立な多重世界の実現法について述べる。

時間独立な多重世界はＡＮＤ関係で結ばれた複数のリテラルの一方で新しい世界を生成し、他方でその世界を指定してゴールを解くという形を考えている。各ゴールフレームにはそのゴールフレームを解くべき世界へのポインタがヘッダとして含まれている。ここで世界とは定義節集合を現わすもので、述語名の識別番号にもとづいたハッシュテーブルおよびハッシュチェインで管理することを考えている。新たな世界を生成する場合には、このハッシュテーブル、および定義を書き換えようとする述語を含むハッシュチェインのうちその述語より前の部分を作りなおさなければならない。通常のプログラムでは動的に定義を変更する述語は限られているので、ハッシュチェインの変更の際にこの述語を先頭に持って来ることにより、次回以降の定義変更の際の書き換え量を減らすことが期待できる。ハッシュテーブルのエントリ数を約100 、ハッシュチェインの長さを数十と仮定した場合に、新しい世界を作る際に必要な書き換えの量は約百数十セルとなり、これはおよそ１ゴールフレーム分に相当する。このようにして作成されたハッシュテーブルなどはＳＭに格納され、あるＩＵにおいて単一化に実際に必要となった時点で、ＳＭからＩＵに対しマルチキャストされる。

また、指定された世界でゴールを解くことはゴールフレームヘッダにつけられた世界を更新することにより実現可能であるが、もとの世界で解くべきゴールが残っている場合には、それらをまとめてsolve の形に直しておく必要がある。以上のように、新しい世界を設定してゴールを解きはじめるまでに、通常のＵＰにおける１回の書き換え量の数倍で実現可能である。

４．おわりに

ＰＩＥにおいて解の寄せ集めおよび時間独立多重世界を実現する方式について述べた。今後これらにつきシミュレーションにより評価を行なうとともに、必要とされる言語機能についても、より詳細な検討を行なっていく予定である。

```
create_world(Def, New_world)
assert_world(Def, [Old_world,] New_world)
retract_world(Def, [Old_world,] New_world)
replace_world(Def, [Old_world,] New_world)
abolish_world(Pred, [Old_world,] New_world)

solve(Goal, World)
```

図２．　時間独立多重世界のための組込み述語

```
assert_world((p:-q), New_world))
```

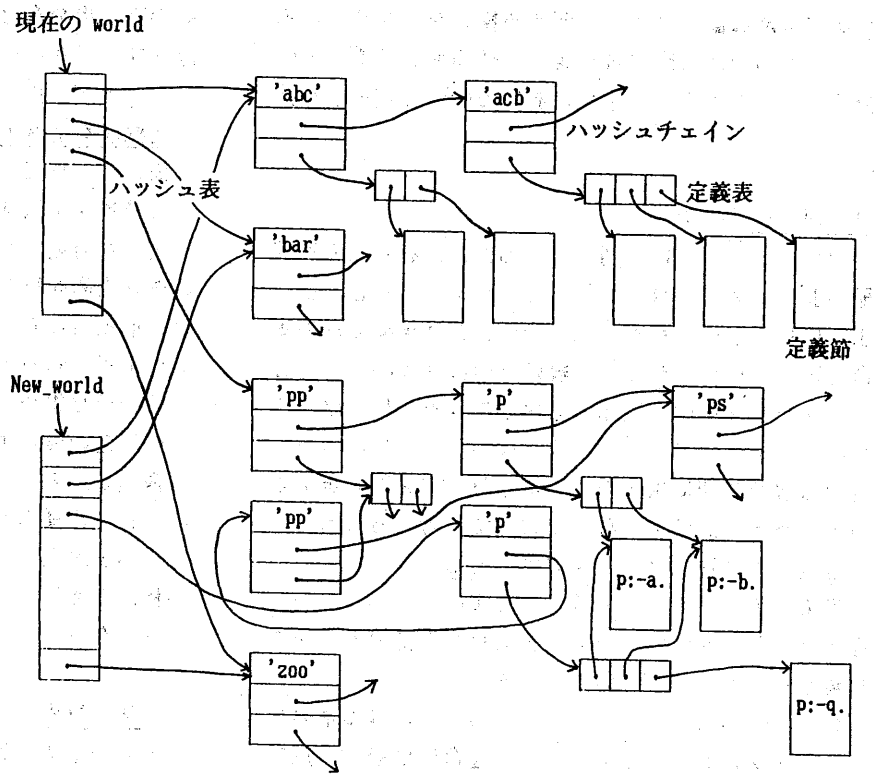

図３．　時間独立多重世界の実現法

```
(w1)?- solve(p, w2), q.
          |
          V
(w2)?- p, solve(q, w1).
```

図４．　ゴールフレームの世界の更新

＜参考文献＞

［１］相田他、「並列処理向き論理型言語の提案」、情報処理学会ソフトウェア基礎論研究会、1984年９月発表予定。