知識工学と人工知能　45－9
（1986・3・10）

プログラミング言語としてのＯＰＳ５

日本アイ・ビー・エム株式会社 サイエンス・インスティチュート　戸沢 義夫

1　はじめに

ＯＰＳ５[1]はプロダクション・システムのルール記述用言語であり、前向き推論を行うエキスパート・システム用の shell として知られている[2]。ＯＰＳ５では、単独で知識を表現しているルールの他に、いくつかのルールがまとまって一つの知識を表現しているものや、通常のプログラミング言語（PASCALやFORTRAN等）の一つの実行文やサブルーチン等のように使われるルールを書くこともできる。これらのルールは単独では知識を表現しているとは言い難いが、これらをうまく使うと自由にプログラムを書くことが可能になる。従って、ＯＰＳ５はプログラミング言語として使用でき他の言語には無い長所を持っている。

ＯＰＳ５の長所は言語が持つプログラミング・パラダイムにあると考えられる。ＯＰＳ５が提供するプログラミング・パラダイムは『トリガーによるプログラミング』である[3]。ＯＰＳ５でプログラムを書いてみると言語の持っている制約（syntax上、vocabulary上、implimentation上）が非常に多く、この制約のために思ったようにプログラムを書きにくいという不満はその通りである。しかし、これらの制約は『トリガーによるプログラミング』というプログラミング・パラダイムを変更することなしに緩和することが可能である。

ＯＰＳ５でプログラムを書こうとしてプログラムが書きにくく感じられるもう一つの原因は、プログラムすべてをトリガーを利用して書かなければならないというプログラミング・パラダイムに起因する。『トリガーによるプログラミング』に対して従来のプログラミング言語は『順次処理によるプログラミング』といえる。明らかに順次処理の方が書きやすいプログラムはたくさんある。しかし、筆者は一つの言語が二つのプログラミング・パラダイム、『トリガーによるプログラミング』と『順次処理によるプログラミング』、の両方を使えるようにする（例えばＯＰＳ８３[4]）よりは、プログラマーに一つのプログラミング・パラダイムを強制した方が良いと考える。ＯＰＳ５はトリガーによるプログラミングを強制するので、それがどのような効果を持つかについて本論文で論ずる。

2　プログラミングとは？

先ず筆者が『プログラミングとは何か』についてどう考えているかを明らかにしておきたい。プログラミングとは、入力が与えられた場合にそれに応じて必要な出力を出すための手順を決めることである。手順とは、個々の実行単位がどのような順序で実行されるべきであるかを決めたものである。個々の実行単位には変数の参照、代入、基本演算、入出力のようなシシテムにとってプリミティブなものからサブルーチンや関数のように抽象化されたものまである。プログラムは、ある入力が与えられた場合にどの実行単位がどの順序で実行されるべきかを記述したものである。従って、プログラミングにとって基本的な作業は次の三つであると考えられる。

（1）実行単位としてどのような抽象化を考えるか。
（2）実行単位をどのような順序で実行するか。
（3）上記をプログラミング言語で記述する。

実行単位の順序の記述法として、従来のプログラミング言語では順序通りに並べて書く方法が採用されている。実行順序通りに並べて記述するのは人間にとって極めて自然でありプログラム・カウンタによって実行が制御される考え方を踏襲している。

一般にプログラムはいろいろな入力を受付けそれに応じた出力を出さなければならない。従って入力の内容に依存して実行される実行単位やその順序も変わる。プログラミングの際には、プログラマーは起り得るすべての入力に対して何がどのような順序で実行されるかを決定している。

筆者はプログラミングに於て実行順序を決定することと、それを記述することとを区別して考える。実行順序の記述法がプログラム・カウンタを仮定して順序通り並べて書くか、トリガー・メカニズムを仮定して puroduction（すなわち ＩＦ ... ＴＨＥＮ ...）で書くかの違いを、本論文ではプログラミング・パラダイムの違いと呼んでいる。前者が『順次処理によるプログラミング』であり、後者が『トリガーによるプログラミング』である。プログラミングに於て実行順序を決定することとその記述とは別であるといっても、実際には実行順序の決定は記述方法や記述能力に大きく左右される。特に実行順序の記述法がプログラム・カウンタを仮定するか、トリガー・メカニズムを仮定するかは非常に大きな違いであると思われるので、本論文で敢えてとりあげるのである。（ＰＲＯＬＯＧは実行順序の記述法にユニフィケーションを仮定するので別なプログラミング・パラダイムを提供していると考えられる。）

３　プログラミングに於ける悩み

本節では筆者がプログラミングに於てどんなことで悩んでいるかを書いておく。大きな悩みの一つはほとんどのプログラムに保守が必要なことである。プログラムを書き始めた時には予想もしなかった要求が後になって発生することはよくあることである。新しい要求を満たすようにプログラムを修正する際には、現在の仕様で正しく動いているプログラムを壊して新しいプログラムを書かなければならないことが多い。せっかく正しく動いているプログラムを一旦壊して動かなくしなければならない時間があるのが気にいらないのである。できれば今動いているプログラムに全く手を加えずに、新しい部分を追加するだけで新しい要求を満たすようにしたい。

もう一つの悩みは、難しいプログラムの場合はどのような手順で何を行えば良いかがわからないことである。例えば、日本語を英語に翻訳するプログラムの場合は、入力とそれに対応した出力の要求を述べること（ある日本語の文が入力された時、それと同じ意味を持つ正しい英語の文を出力する）は簡単であるが、実際にどのような日本語の文が与えられた場合でも翻訳できるようにするにはどうしたら良いかわからない。しかし、一般的な翻訳プログラムは難しいとしても、入力される文が特定の条件を満たす場合に限られているならば処理手順を決められる場合がある。難しいプログラムの場合に、汎用性を減らして特殊な場合にだけ動作するプログラムを先ず作り、後から汎用性の要求を追加してプログラムがだんだん intelligent になっていくようなプログラミングを行いたい。この場合でも、特殊な場合にだけ動く最初に作ったプログラムに手を加えずに新しい部分だけを追加してプログラムをより intelligent にしたいのは上と同じである。

良いプログラム作成法とは、先ず要求分析をして仕様を決め、その後設計をしてプログラムを書くべきであると主張されている。確かにこのようなアプローチをとると正しく仕様を満たすプログラムを確実に作ることができる。しかし、要求が変化し仕様が変わるような場合に生じるプログラミング上の悩みはこのアプローチでは解決しない。特に、このアプローチで問題だと思われるのは完全に仕様を満たすプログラムが完成するまでプログラムが動かない点である。つまり、仕様を80%満たす程度にプログラムが完成している段階ではプログラムは動かないのである。筆者が求めているのは、仕様を20%しか満たさない場合はそれで動き、50%になれば50%で動き、80%,100%とだんだん機能や処理能力が増えていくようなインクリメンタルなプログラミングである。そして、機能や処理能力を増やす部分は既にある部分に全く手を加えずに後から追加するだけで済ましたいのである。

４　トリガー・メカニズム

本節ではＯＰＳ５で使われている概念を示し、どのようにしてトリガー・メカニズムが実現されているかについて述べる。ＯＰＳ５で使われる主な概念には次のようなものがある。

(1) Working memory と Working memory element
(2) Production memory と Production
(3) パターン・マッチ
(4) Conflict Set と Instantiation
(5) Conflict resolution strategy

4．1 Working memory と Working memory element

Working Memoryはデータを格納する場所である。データはWorking Memory Elementとして格納される。(以後Working Memory ElementをＷＭＥと略す。) 各ＷＭＥはクラス名といくつかの属性を持つ。次にＷＭＥの例を示す。

(社員 ↑姓 戸沢 ↑名 義夫 ↑性別 男 ↑部門 研究所 ↑婚姻 既婚)

社員はクラス名、↑は属性を識別するためのオペレータである。姓、名、性別、部門、婚姻は属性名であり、戸沢、義夫、男、研究所、既婚が各属性の値である。

Working Memoryへアクセスする方法は次の４通りである。

(1) make 新しいＷＭＥを Working Memory へ追加する。
(2) remove ＷＭＥを Working Memoryから削除する。
(3) modify ＷＭＥの値を更新する。
(4) 参照 ＬＨＳの条件要素とのパターン・マッチによる。

4．2 Production memory と Puroduction

Production memory はProduction を格納する場所でありプログラムの実行部である。Productionは ＩＦ 条件部 ＴＨＥＮ 実行部 の形をした規則を (p production名 条件部 --> 実行部)の構文で書いたものである。条件部をＬＨＳ(left hand side)といい実行部をＲＨＳ(right hand side)という。Productionは同時には１つだけが発火(fire)しそのＲＨＳが実行される。発火可能なproductionがある限り次々と発火し続けプログラムの実行が行なわれる。次にproductionの例を挙げる。

```
(p 社内結婚
{<male>  (社員  ↑性別  男  ↑婚姻  独身  ↑部門  <d>  ↑姓  <name>) }
{<female> (社員  ↑性別  女  ↑婚姻  独身  ↑部門  <d>) }
-->
(modify <male>    ↑婚姻  既婚)
(modify <female>  ↑婚姻  既婚  ↑姓  <name>) )
```

この production は同一部門に独身の男性と女性がいると、その二人は社内結婚をし女性の姓が男性の姓に変わることを示している。<name>,<d>はＷＭＥの属性値の参照に使われる変数であり、<male>,<female>は要素変数といい {}と組合せてＷＭＥそのものの参照に使われる。Production Memoryに許されている操作は、新しい production を追加(既に production名の同じものがあるときは変更)する build だけである。

4．3 パターン・マッチ

Productionの条件部（ＬＨＳ）は Working Memory の内容と照合される。この照合は多対多のパターン・マッチである。照合の結果、条件部が満たされていると、その production は発火可能である。条件部は条件要素の並びであるので、すべての条件要素が満たされている（真である）場合に条件部

は満たされる(真である)。ある条件要素は、それとパターン・マッチするＷＭＥが Working Memory 内に存在する(否定条件要素の場合は存在しない)時に真になる。

条件要素に書かれた変数はすべての値とパターン・マッチする。その結果、変数はパターン・マッチしたＷＭＥが持つ値にbindされる。条件部に同一変数が現れている時はそれらの値は同じでなければならない。例えば、前節の'社内結婚'の中で使われている<d>は最初の条件要素と二番目の条件要素で同じ値でなければならず、『同一部門』という条件を表わすことになる。

４．４　Conflict Set と Instantiation

発火可能なproductionは条件部を満たしているＷＭＥと組にされ、１つのinstantiationを形成する。あるproductionの条件部を満たすＷＭＥが複数個あったり、あるＷＭＥが複数のproductionの条件部を満たしたりすると、instantiationは２つ以上になる。Instantiationの集合をConflict Setという。例えば Working Memory に次のようなＷＭＥがあるとする。

w1　(社員 ↑姓 佐藤 ↑性別 男 ↑部門 SE ↑婚姻 独身)
w2　(社員 ↑姓 鈴木 ↑性別 男 ↑部門 SE ↑婚姻 独身)
w3　(社員 ↑姓 渡辺 ↑性別 女 ↑部門 SE ↑婚姻 独身)

'社内結婚'を満たすＷＭＥは２組あるので Conflict Setには次の２個のinstantiationが含まれる。

ins1　(社内結婚 w1 w3)
ins2　(社内結婚 w2 w3)

４．５　Conflict Resolution Strategy

一度に発火できる productionは一個だけなので、発火可能な instantiationの中でどれが実際に発火するかを決めなければならない。Conflict Setの中から実際に発火する instantiationを１つ選び出す方法を Conflict Resolution Strategy(又は単にstrategy)と言う。一般に strategyが異なるとProduction Memoryと Working Memoryの内容が同じ状態から出発しても結果が異なってしまう。

ＯＰＳ５で提供されている strategyにはＬＥＸとＭＥＡがある。これらの strategyではＷＭＥが作られた(makeされた)時刻の最新さを示す recency が重要な役割を果たす。これらの strategy で使われている基本的な考え方は次のようである。

（１）一度発火に使われた instantiation は二度と発火しない。
（２）最新のＷＭＥを含む instantiation が選ばれる。
　　(recency の扱い方はＬＥＸとＭＥＡで少し異なる。)
（３）Recency の同じＷＭＥが複数の production の条件部を満たしている場合は、条件部がより specific な(条件がきつい) production を含む instantiation が選ばれる。

ＯＰＳ５をプログラミング言語として使用する場合には、あるproductionの次にどのproductionが発火すべきかをプログラーがきちんと規定できないと困る。この点でＭＥＡの方が優れておりプログラミングにはＭＥＡを用いるのが良い。ＭＥＡは次のように定義されている。

(1) 既に発火に使われたinstantiationを対象から外す。
(2) 条件部に書かれている最初(一番左)の条件要素にマッチしているＷＭＥのrecencyを比べ、最も最新のＷＭＥを含んでいるinstantiationを選びだす。Instantiationが一つなら終了する。
(3) (2)の結果二つ以上のinstantiationが残った場合は、各instantiation毎に、二番目以降の条件要素にマッチしているＷＭＥをrecencyの順にsortする。各instantiationからsortされたＷＭＥの

最新のものを取り出してrecencyを比較する。古い方のinstantiationは外される。これで決着が着かなければ次に最新のものを取り出して比べる。WMEが取り出せなくなったinstantiationは外される。これを決着が着くまで繰り返す。

(4) (3)でWMEが同時に無くなってinstantiationが一つに決まらない場合は、(3)で最後まで残ったinstantiationのproductionの条件部を比較する。条件部がパターン・マッチするのに必要なテスト(比較)の回数を数え、最もテスト回数の多いinstantiationが選ばれる。

(5) (4)で決着が着かない時はくじ引きによる。

４．６　トリガーとその実現メカニズム

Productionは　ＩＦ　条件部　ＴＨＥＮ　実行部　の形をしている。これを普通に解釈すると『条件部が真であれば実行部を実行する』となる。しかし、ＯＰＳ５では strategy の中に『一度発火したinstantiationは二度と発火しない』という規則が含まれているため、productionの解釈は『条件部が偽から真になれば実行部を一回だけ実行する』となる。これがトリガーである。

条件部はパターン・マッチによりWorking Memoryと照合されるので、ＷＭＥが追加(make)されたり削除(remove)されたりする度に条件部が真になるproductionをトリガーする。トリガーされた productionは発火可能な productionがいくつもあると発火まで待たされることがある。待っている間に条件部が真でなくなり Conflict setからはずされトリガーされても発火しないで終わることもある。

以上述べてきたことをまとめると図１になる。ＯＰＳ５ではパターン・マッチにＲＥＴＥアルゴリズム[5]を用いてトリガー・メカニズムを実現している。これは図１に示すように Production Memoryと Working Memoryを照合して Conflict Setを計算する際に使われる。一つの productionの発火によって変更される Working Memoryは全体のごく一部に過ぎないという性質を利用し、パターン・マッチ計算の中間結果をネットワークのノードの中に保存しておく。これにより実際に行われる照合計算は Working Memoryの変更分だけで済むので計算量が減り高速化される。

一つの productionは一般に図２の形のネットワークになる。複数のproductionに共通なノードは共用される。αメモリのノードは入力が一つであり、βメモリのノードは入力が二つである。αメモリのノードには個々の条件要素を満たすＷＭＥが格納され、βメモリのノードには二つ以上の条件要素を満足するＷＭＥの組が格納される。βメモリでのネットワークの形は条件要素が書かれた順に左から右へ評価されていくことに対応している。

Working Memoryに変更(make又はremove)が生じると rootノードに makeか removeかのタグの付いたＷＭＥがトークンとして渡される。このトークンはネットワークを通りながら、消滅したり新しいトークンを発生したりして Conflict Setへと向かう。Makeのトークンが Conflict Setまで辿り着くと instantiationを形成し productionをトリガーする。Removeのトークンが Conflict Setに辿り着くと、対応する instantiationは Conflict Setから外される。トークンが途中で消滅した場合はConflict Setから instantiationが選ばれて発火する。発火した instantiationは Conflict Setから外される。ＲＥＴＥアルゴリズムを使用する場合は Conflict Setは Working MemoryとＬＨＳの照合結果ではなく、トリガーされたが発火していない instantiationの集合として実現されている。

５　トリガーによるプログラミング

５．１　作業順序の規定法

ＯＰＳ５では作業Ａと作業Ｂの順序を規定する書き方が二通りある。一つの書き方は次の様に実行順序(の逆)通りに並べて書く。

```
(p 仕事
  ( ... )
-->
```

```
  (make 作業 B)
  (make 作業 A))
```

WME(作業 A)と(作業 B)はそれぞれ作業Aと作業Bをトリガーするものとする。このproductionが発火すると作業Aと作業Bの両方がトリガーされる。WME(作業 A)はWME(作業 B)の後に作られるので、(作業 A)のrecencyが高くなり作業Aが作業Bよりも先に行なわれる。作業Bの実行は作業Aの実行によりトリガーされるすべてのproductionの発火が終るまで待たされる。なぜなら、作業Aでトリガーされて作られる instantiationの recencyがWME(作業 B)の recencyよりいつでも高いからである。

作業Aの次に作業Bが行なわれる事を規定するもう一つの書き方は次のようである。

```
(p A終了後B
  (作業 A)
-->
  (make 作業 B))
```

Production 'A終了後B' はWME(作業 A)によって作業Aを実行するためのproductionと同時に発火可能になる。しかし、strategyにより条件部がより specificな(条件がきつい)productionが先に実行されるため、'A終了後B' は作業Aの実行によってトリガーされる productionがすべて発火し終わるまで発火しない。'A終了後B' の条件部はWME(作業 A) によってトリガーされるどのproductionの条件部よりもゆるいからである。作業Aの実行によりトリガーされる instantiationのrecencyはWME(作業 A)の recencyよりも高いので、作業Aがすべて終了してから 'A終了後B' が発火しWME(作業 B)が makeされる。

後者の書き方のように作業Aと作業Bの順序関係の記述が一つのproductionで規定できると、作業順序の入れ替えや、新しい作業の挿入などに対して極めて柔軟性が出てくる。つまり、プログラムの変更が作業順序の入れ替えや新しい作業の挿入などの場合は、作業の実行順序を制御している productionを変更するだけで良く、現在動いているプログラムには全く手を加えずに行えるのである。

５．２ モニター機能

トリガーによるプログラミングでは、どこでいつ発生するかわからない事象をモニターして対応策を講じるようなプログラミングが簡単に書ける。ＹＥＳ／ＭＶＳ[6]がＯＰＳ５を使用したエキスパート・システムとして成功したのは、問題の性質（ＭＶＳオペレータ作業の自動化）がモニター機能によって解き易かったからである。このようにモニター機能を使用すると解き易くなる種類の問題はかなり多いと思われる。

プログラムの仕様が変わった場合に、新しい仕様の影響を受ける部分だけをモニター機能を用いて追加することが可能である。次にモニター機能として書かれた productionの例を挙げる。

```
(p 温度検査
  (観測値 温度 >= ６０)
-->
  (make 高温処理))
```

温度が60度以上の時には高温処理をするように仕様が変更された場合に、高温処理のプログラムを書き、'温度検査' の productionを追加するだけで、もとのプログラムに全く手を加えずに新しい仕様を満たすことができるのである。

５．３　トラップ

トリガーによるプログラミングではプログラマーが productionの発火順序を決めているのであるが、本来意図した productionが発火する前に別なproductionを発火させてプログラムを迂回させることができる。これががトラップである。Productionがいつ発火するかはプログラム全体を見なくても puroductionの条件部を見るだけでわかる。条件部を満たすＷＭＥの recencyが同じ時は条件がより specificなものが先に発火するので、トラップをかけたい productionの条件部にトラップ用の条件要素を追加した productionを書くことにより、任意の productionにトラップをかけられる。

トラップの考え方を図示すると図３のようになる。『本流』は問題の基本部分を解くためのもので、『副流』は問題の細部を詰めるためのものである。問題を基本部分と細部とに分けて考え、基本部分のプログラミングを先に行い、後で細部のプログラミングを行う。トラップを利用すると、基本部分のプログラムに全く手を加えずに、細部のプログラムを追加することが可能になる。

このアプローチをとると細部の設計が完了する前にプログラムを書き始めることができ、どのような設計が良いかがはっきりわかっていない場合に強力である。また、新たなrequirementがわかってシステムの変更が必要になる場合でも保守が容易になる。

６　まとめ

ＯＰＳ５をプログラミング言語として使った場合の最大の特徴は『トリガーによるプログラミング』を強制される点にある。プログラマーがある要求仕様に基づいてプログラミングした際に決めた手順は、プログラムの仕様が変わったりプログラムに対する要求が一般化されたりすると変更されなければならなくなる。要求や仕様が変わった場合に、新しい要求仕様を処理する部分だけをプログラミングして、元のプログラムに全く手を加えずに追加することで対処したい。つまり、インクリメンタルにプログラミングを行うことを考える。

元のプログラムで決められている手順を変更して新しい部分を追加する方法には次の３通りある。

（１）作業順序を規定している部分を書き換えて新しい部分を追加する。
（２）ある部分の実行が終了した後に新しい部分が実行されるようにする。
（３）ある部分の実行が始まる直前に新しい部分が実行されるようにする。

ＯＰＳ５でプログラムが書かれていると、作業順序を明示的に記述する部分は5.1節で示した二番目の書き方のように前後関係だけを示す簡単な puroductionになる。従って上記(１)を行うにはこのproductionを書き換えるだけで良く、元のプログラムのそれ以外の部分には手を加えずに行える。上記(２)はモニター機能で対処できる。特に『ある部分』がプログラム内のあちこちに散らばっている時などでも一つの productionの追加で対処できるのは強力である。上記(３)はトラップである。モニター機能もトラップも元のプログラムに全く手を加えずに productionの追加だけで行える。このように要求仕様が変わってもインクリメンタルにプログラミングできるのがプログラミング言語としてのＯＰＳ５の特徴であり『順次処理によるプログラミング』では実現困難な長所である。

インクリメンタルにプログラミングできるのは『トリガーによるプログラミング』に起因する長所である。『トリガーによるプログラミング』は『順次処理によるプログラミング』を一般化したものと考えられる。ＯＰＳ５ではトリガーを実現するメカニズムとしてＲＥＴＥアルゴリズムが使われている。プログラムの実行効率はＲＥＴＥアルゴリズムがどれだけオプティマイズできるかで決まる。[３]で述べたようにＯＰＳ５には言語の定義に syntax上、vocabulary上、implimentation上の制約がありプログラミング・パラダイムと相入れない部分もある。

プログラミング言語としてどのようなものが良いのかについてはいろいろな議論があると思われるが『トリガーによるプログラミング』という新しいプログラミング・パラダイムを提供するＯＰＳ５は、エキスパート・システムの shell としてだけでなく新しいプログラミング言語の方向を示すものと考えて良いと思われる。

参考文献

【1】Forgy, C. L., "OPS5 User's Mannual", Technical Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, July 1981

【2】Brownston, L., Farrell, R., Kant, E. and Martin N., Programming Expert Systems in OPS5, Addison-Wesley, 1985

【3】戸沢 義夫、"ＯＰＳ５が提供するプログラミング・パラダイムと実行効率について",第２７回プログラミング・シンポジウム報告集, pp 1-12, 1986

【4】Forgy, C. L., "The OPS83 Reference Manual" Department of Computer Science, Carnegie-Mellon University, January 1984

【5】Forgy, C. L., "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem" Artificial Intelligence 19(1), September 1982

【6】Griesmer, J., Hong, S., Karnaugh, M., Kastner, J., Schor, M., Ennis, R., Klein, D., Milliken, K. and Van Woerkom, H., "YES/MVS: A Continuous Real Time Expert System," Proc. of American Association of Artificial Intelligence, pp 130-136, 1984

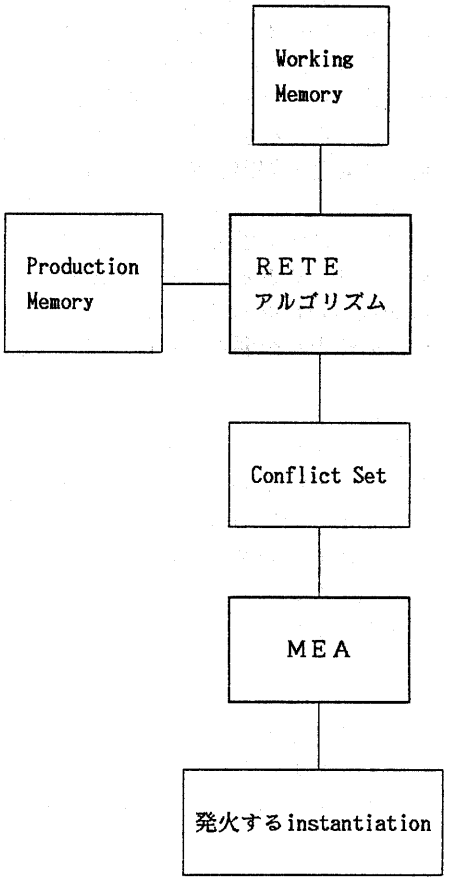

図１　発火するInstantiationが選ばれるまで

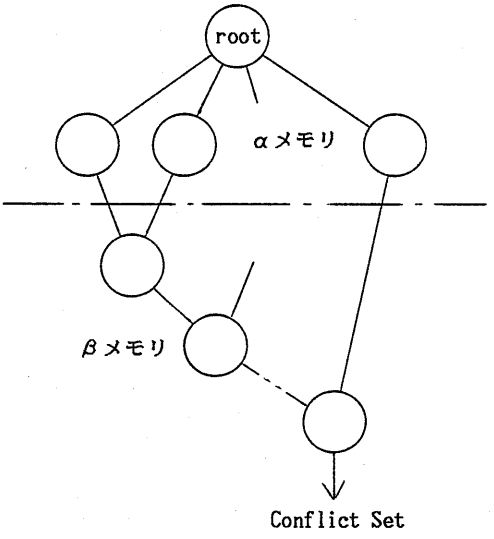

図２　ＲＥＴＥネットワーク一般形

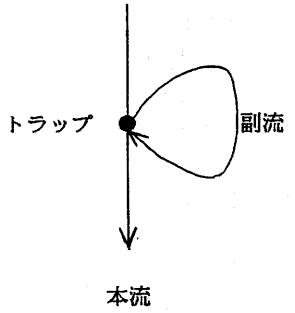

図３　トラップの考え方