情報処理学会第37回(昭和63年後期)全国大会

5J-4

プロダクションルールシステムにおける不完全述語処理

髙野　裕之　　横山　貴子

日本電気ソフトウェア株式会社

1. はじめに

プロダクションルールシステムにおいて、ルールは一般に「もし　<条件部>　ならば　<結論部>」の形式で記述される。この<条件部>には比較式などの、ルールが成り立つための条件を記述する。通常、比較式は「変数　比較演算子　定数」の形をしている。しかしながら、このような比較式だけでは条件を一つ一つすべて記述しなければならないため、柔軟な知識処理をすることができない。このため比較式だけではなく、述語表現をプロダクションルールに採り入れ、推論時にワーキングメモリ上に生成した述語と、条件部の述語とのマッチング処理を加えることによって柔軟な知識表現ができるようにする。

述語表現の特徴は、述語変数を用いて一つの述語表現が複数の述語とマッチすることができる点である。本稿では、述語表現を採り入れたときの手法として、バックトラッキング時の戻り位置を記憶しておくためのアドレススタックを用いた、述語変数の処理についてC言語上での実現した結果を述べる。

2. 概要

述語表現を実現するときに問題となるのが述語変数の処理である。なぜなら述語変数を導入することによって、条件部のチェックは一意に決定しなくなり、一つの条件部に対して述語変数の与え方により、何通りかの組合せで条件部が成立することになる。例えば、条件部に不完全述語（述語変数を含む述語）が2つあり、ワーキングメモリに述語が6つ生成されているとき、この条件部とワーキングメモリの述語との組合せは全部で6^2＝３６通りある。この３６通りの組合せについてすべて条件部をチェックし、成立する組合せのみを抽出する必要がある（図１参照）。これは、３６個のルールを記述したものとほぼ同等となる。

従って、述語表現（述語変数）をプロダクションルール上で実現するには、すべての条件に対していかに効率よく、もれなくチェックを行うかが特に重要となってくる。

```
ルール
もし
    （症状　氏名　発熱）；　……①
    （性別　氏名　男）；　　……②
ならば
    ～～；
```

”氏名”が述語変数とする

```
ワーキングエリア
（症状　山本弘　発熱）　　……a
（症状　田中次郎　頭痛）　……b
（症状　鈴木純子　発熱）　……c
（性別　山本弘　男）　　　……d
（性別　田中次郎　男）　　……e
（性別　鈴木純子　女）　　……f
```

この例では、３６通りの組合せのうち「①－ａ，②－ｄ」の１通りが成立する。

図１　述語のマッチング例

本稿で述べる手法の特徴としては、次の３つを挙げることができる。

（１）C言語上での実現を目的としており、C言語の高速性を利用できる。

（２）述語のために特に複雑な処理を必要としない。

（３）述語を処理することで、推論速度を大きく低下させることがない。

3. 手法

推論の前提としてルールは、あらかじめすべて構文解析を行い、解析結果を木構造に変換しておく。そして推論時は、この木構造を用いるものとする。またここでの説明は、条件部に不完全述語のみが記述されているものとし、説明のために条件部の述語を①、②、…で表し、ワーキングメモリ上の述語をａ、ｂ、…で表すことにする。

条件部のチェックは①に対してａ、ｂ、…と順に調べていく。同様に②に対してａ、ｂ、…と調べていくが、このとき①をチェックした結果の述語変数の値（述語変数の拘束状態）を考慮しなければならない。

Implementation of Incomplete Predicate on Production Rule System
Hiroyuki TAKANO, Takako YOKOYAMA
NEC Software, Ltd.

次に、条件部チェックに必要な仕組みを述べる。

（１）アドレススタック

ある述語のチェックをしたときに、チェックしている述語が成り立ったならば、その述語の木構造でのアドレスをスタック上に積む。これは、述語のすべての組合せを調べるために必要な、バックトラッキングの戻り位置となる。

（２）評価コード

バックトラッキングのときには、条件部のチェックをもう一度最初の処理点から行わなければならない。これは、処理点を任意の位置に戻すには、ローカル変数等の情報を逐次保持しておかなければならないため、処理が複雑になるからである。しかし、一度チェックしたところもすべてチェックしなおしたのでは推論速度の低下を招くことになる。そこで一度チェックした条件で、不確定要素のないところ（不完全述語を含まないところ）は、チェックを繰り返さないようにするために評価コードを用いる。この評価コードには、各条件のチェックした結果を保持しておく。

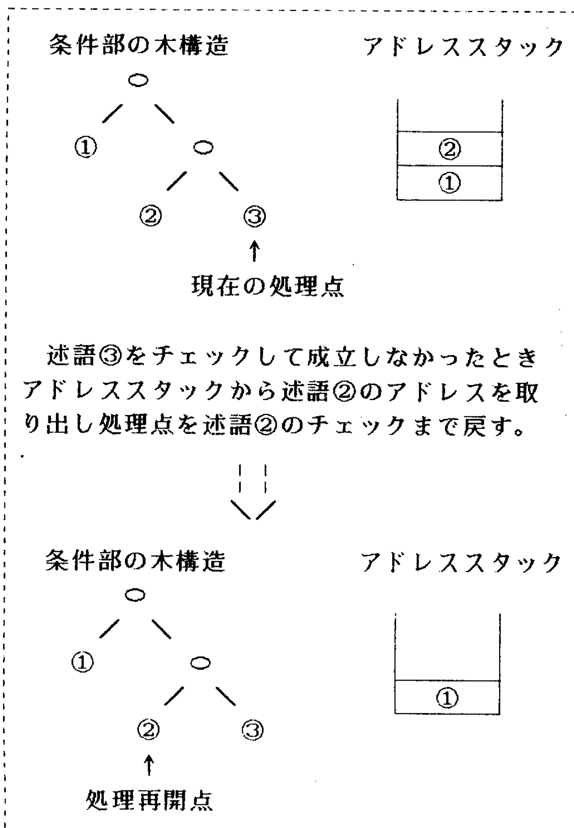

図２　条件部のチェック

（３）述語変数テーブル

述語中に含まれる述語変数を、このテーブルで各ルール毎に管理し、推論時の変数の拘束状況を把握する。

これら３つの項目を加えることによって、プロダクションルール中に述語表現を採り入れることができる。

以下に推論処理をステップ毎に示す。

1.Ｃ言語のsetjmp関数を用いて条件部チェックの処理開始点を保存する。
2.条件部の木構造をたどり述語が成り立っているかどうかをチェックする。
3.述語が成り立っていればアドレススタックに、木構造における述語のアドレスを積む。
4.条件部全体が成立したときは、それを記録しステップ5.から実行する。
5.アドレススタックを参照し要素があれば、１つ取り出してバックトラッキングの戻り位置として設定する。
6.Ｃ言語のlongjmp関数を用いて処理点を処理開始点に戻して、アドレススタックから取り出した戻り位置のところまで、疑似的に条件部のチェックをしなおす。
7.アドレススタックから取り出した戻り位置のところより条件部チェックを再開する。
8.ステップ5～7をアドレススタックの要素がなくなるまで繰り返すことにより条件部の述語とワーキングエリアの述語の全ての組合せについて調べることができる。

４．おわりに

本稿で述べた手法では、推論速度を極端に低下させることもなく述語表現をプロダクションルール上で実現することができた。また述語表現の実現により、推論時に試行錯誤を必要とするような、設計型の領域に対してもプロダクションルールを用いることができるようになった。

今後、この手法でより高速に処理するためには、ワーキングメモリ上の述語をソートし、条件部のチェック時には全ての組合せをチェックしないようにすることなどが考えられる。

参考文献

［１］高野：後ろ向き推論の一方法について，情報処理学会第３６回全国大会，４Ｒ－５，（１９８８）