情報処理学会第29回(昭和59年後期)全国大会

ＰＩＥにおける構造メモリの構成法

2B-3

平田　圭二、　田中　英彦、　元岡　達

（東京大学　工学部）

１．はじめに

高並列推論エンジンＰＩＥにおいて構造データを共有する方式を採用することで、１桁程度の高速化が達成可能であることはシミュレーションによってすでに確認されている［１］。テストプログラムでは、ゴールフレーム（ＧＦ）のデータ長が約１／２に短縮され、縮退に要する時間も約１／２に減少した。実際の応用プログラムではＧＦがさらに大きな構造データを抱えることが予想されるが、本方式では演算時間を構造データ全体の大きさに直接関係しないようにすることができる。このような演算機構を効率良く実現するため、我々は現在ＰＩＥのネットワークを階層化し、構造データを格納する構造メモリ（Structure Memory，SM）を設けることを検討している［２］。

本発表ではＧＦの内部表現、単一化・縮退アルゴリズム、SMに要求される機能について述べる。

２．ＧＦの内部表現と各方式

２．１　共有について

これまで検討してきた構造データの共有方式では未定義変数を含まない構造データ部分（Ground Instance，ＧＩ）をＧＦ間で共有していた。この方式は読み出しのみ行い、書き換えの生じないＧＩの部分だけを共有するので、ＧＦ間の論理的独立性を高く保つことができる。しかし実際の応用プログラムを観察すると、次のような構造データが頻繁に出現することがわかる（図１）。

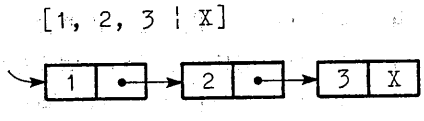


図１．　未定義変数を含む構造データ

この場合、先頭２セルには未定義変数が含まれないが、３セルめの未定義変数Ｘの為に、全体が共有の対象とはなり得ない。従って、構造データの共有度理の妨げとなることに注意しなくてはならない。以下に構造データの共有度が異る３つの方式を示す。方式１から方式３の順で共有度が上がるにつれて、ＵＰ側での処理負荷は減り、より高い機能を持ったSMが必要になる。

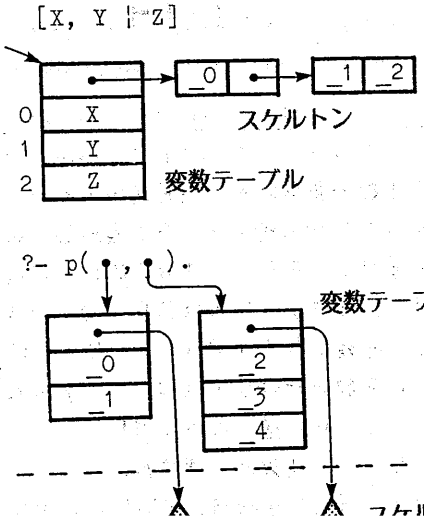
２．２　方式１：Ground Instance（ＧＩ）だけの共有方式

現在まで提案、検討してきた構造データ共有方式は、ＧＩの部分のみ共有しているので、最も共有度の低い基本的な方式である［１］。

共有の対象となるＧＩの部分はSMに格納され、共有の対象でないＧＦのヘッダ部、リテラル部、未定義変数を含む構造データ部はMMに格納される。この時ＵＰはＧＩ部分に対する追加読み出し操作（Lazy Fetch，ＬＦ）を必要とする。また縮退の際、構造データに対してMMに送るべき部分か、SMに送る部分かの切り分けを施す必要がある。

２．３　方式２：変数テーブルとスケルトンによる構造データ表現

未定義変数を含む構造データをＧＦ間で共有する為に構造データの枠組み（Skeleton，スケルトン）と変数テーブル（Variable Table，ＶＴ）の対によって構造データを表す（図２）。リテラル部から指されるＶＴはＧＦの一部として転送され、そこに



はＧＦ内での変数の通し番号が書き込まれている。ＳＭに格納されるスケルトンは初期ＧＦが読み込まれた時に生成され、以降共有の対象となる。スケルトンの中での変数番号はそのルートから広がる構造データの中で唯一になるようにつけられる。

ＧＦ中の未定義変数を含む構造データ部分を転送するかわりに、ＶＴの部分だけ転送すれば良く、縮退時に書き換えるのもＶＴだけで良い。しかしＬＦを行いスケルトンをたぐり、その一部分しか指さないような状況になっても、ＶＴは縮小できない。また変数の値の束縛によって余分なＶＴの領域が生じることもある（図３）。本来ならば、これら無駄な部分を除去するのが縮退の機能であった。これに対して、ＳＭに未定義変数への値の書き換えを伴うコピーコマンド（リダクションコマンド）を発行することで、縮退と等価な操作をＳＭ内で実行することができる（図４）。

```
Goal : ?- p(X, [1, 2 | X]).

Def. :   p([3, 4, | X], Y) :-  . . .
```

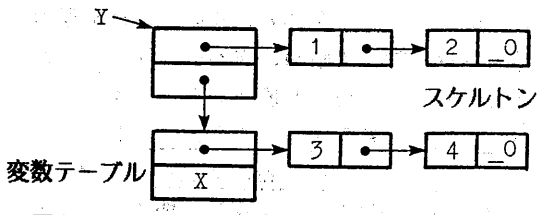


図３．　単一化後の構造データ例

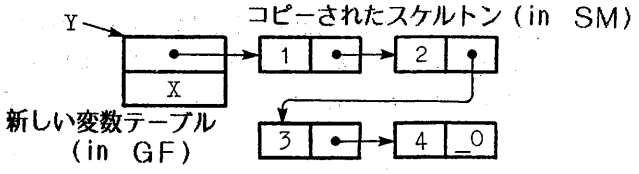


図４．　コピーコマンドによるSM内縮退の様子

２．４　方式３：リテラルレベルからの共有

実際のプログラムではＡppend やＲeverseがあると、リテラル数の非常に多いＧＦが出現する。そこでリテラルレベルから構造データで表現し、単一化に必要な先頭リテラルまでの部分のみＵＰに取り込むことにする。変数テーブルはＧＦ中の全変数分だけの大きさが必要になる（図５ａ）。単一化時、新に導入された変数分だけＶＴは伸び、生成された束縛情報はあとでＳＭに送られる（図５ｂ）。従ってＬＦはスケルトンと変数の束縛情報に対して行われるので、方式２と同様なコピーコマンドを発行し縮退と等価な操作を実現する必要がある（図５ｃ

```
?- app([4, 3, 2], [1], X), pr(X).
```

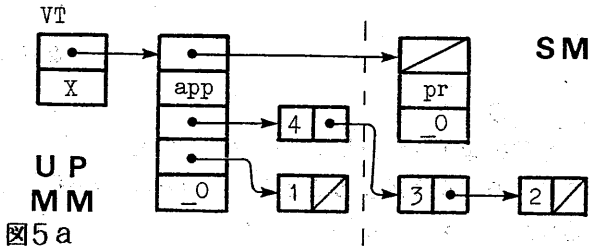


図５ａ

```
?- app([3, 2], [1], Y), pr([4 | Y]).
```

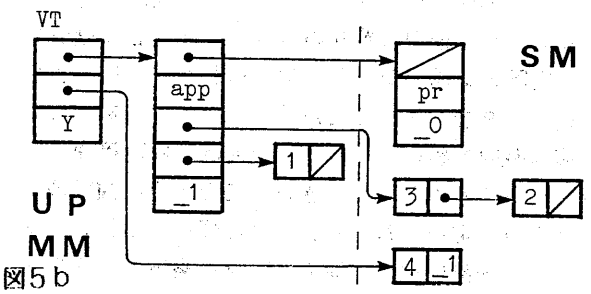


図５ｂ

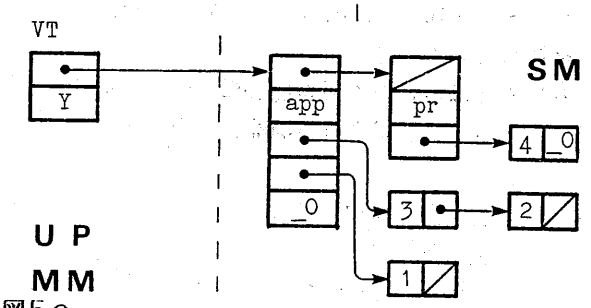


図５ｃ

図５．　方式３での単一化・縮退の様子

３．構造メモリ（ＳＭ）に要求される機能

これまで述べたようにＵＰでの処理負荷、特に縮退の手間が減少する分、ＳＭに処理負荷のしわ寄せがきている。方式１～３の内、いずれの方式の効率が良いかはプログラムの特性によるので一概には言えない。共有度を動的に変化させ、処理負荷のバランスが最良の点で常に動作させるのが望ましい。

またＳＭに対するコマンドとしては、通常のリンクデータ構造に対する演算子の他に、以下のものを用意すれば良いことが分かった。

① 追加読み出し（ＬＦ）
② 変数の束縛を伴なうコピーコマンド

尚ここでは触れなかったがＳＭに関する他の検討