LMNtalルールコンパイラにおける内部命令の設計

水野 謙　永田 貴彦　加藤 紀夫　上田 和紀
Ken MIZUNO　Takahiko NAGATA　Norio KATO　Kazunori UEDA
早稲田大学理工学部
Science and Engineering, Waseda University
mizuno@ueda.info.waseda.ac.jp

LMNtal は階層グラフ構造の書き換えに基づく並行言語モデルである．我々が実装した LMNtal 処理系は，プログラム中のルールを内部命令列に変換するルールコンパイラと，それを解釈実行する実行時処理系からなる．本発表ではルールコンパイラの動作手順およびそこで使用される内部命令について説明する．ルールの実行はマッチングテストと書き換えの実行からなる．はじめに単純な方法でこれらに対応する内部命令列を生成する方法を述べる．次に，この命令列に対して冗長性を削減することによって最適化を行う様子を紹介する．また，我々が設計した内部命令セットが，膜を含まないルールの動作を記述するために十分であることについても触れる．

1　はじめに

LMNtal [1] は大規模計算から極小計算までを簡潔かつ統一的に扱うことを目的とした，多重集合書換え言語の一種で，多重集合が膜によって構造化される，膜が局所的な書換えルールを持てる，およびルールセットが移送可能であるなどの特徴を持つ．

我々が実装した LMNtal 処理系では，プログラム中のルールはルールコンパイラによって内部命令列にコンパイルされ，実行時処理系によって実行される [2]．例えば**図 1** のプログラムは**図 2** と**図 3** の命令列にコンパイルされる．変数 0 にはルールのある膜が渡される．

本発表ではルールコンパイラの動作手順およびそこで使用される命令セットについて説明する．

以下，2 節では命令セットを説明し，3 節ではルールに対して簡単な方法で命令列を生成する方法を述べる．4 節で最適化の方法を説明し，5 節でまとめる．

2　命令セット

内部命令列の命令セットを**図 4** に示す．

deref 命令は，アトム *srcatom* の第 *srcpos* 引数の接続先のアトムを取得し，接続先の引数位置を確認する．

findatom 命令は，指定されたファンクタを持つアトムを順次取得する．これに対し func 命令は，アトムが指定されたファンクタを持つことを確認する．

newlink/inheritlink/unify の各命令はリンクのつなぎ替えに使用する（3.2 節）．

3　ルールコンパイラの動作手順

ルールの実行は，ルールの適用可能性を検査するマッチングと，実際にルールを適用するボディ実行からなる．本節では，それぞれに対応する命令列の部分（図中では点線で区切られている）の生成について述べる．

3.1　マッチングのコンパイル

アトムの検査には，すでに見つかっているアトムとリンクで接続しているアトムを，deref 命令と func 命令を用いて優先的に検査する．それ以外のアトムは，findatom 命令と neqatom 命令を用いて検査する．

膜の検査も同様で，すでに見つかっているアトムから特定可能な膜を優先的に取得し，検査の効率化を図る．

3.2　ボディ実行のコンパイル

ボディ実行では，まず左辺にマッチしたプロセスを除去し，次に右辺に対応するプロセスを生成する．アトムの除去・生成は removeatom/newatom 命令で行う．

リンクのつなぎ替えは次の 3 種類の命令で行う．

- 右辺に 2 回出現するリンクに対しては，newlink 命令を用いる．
- 右辺に 1 回出現するリンクのうち，‘=’ アトム以外

```
( append(X0, Y, Z), cons(A, X, X0) :- cons(A, X1, Z), append(X, Y, X1) ),
( append(X , Y, Z), nil(X)         :- Z=Y )
```

図 1: LMNtal プログラムの例

```
findatom   [1, 0, append_3]
deref      [2, 1, 0, 2]
func          [2, cons_3]
-----------------------------
removeatom    [1]
removeatom    [2]
newatom    [3, 0, cons_3]
newatom    [4, 0, append_3]
getlink    [5, 1, 1]
getlink    [6, 1, 2]
getlink    [7, 2, 0]
getlink    [8, 2, 1]
inheritlink   [3, 0, 7]
newlink       [3, 1, 4, 2]
inheritlink   [3, 2, 6]
inheritlink   [4, 0, 8]
inheritlink   [4, 1, 5]
```

図 2: 1 つ目のルールに対する命令列

```
findatom   [1, 0, append_3]
deref      [2, 1, 0, 0]
func          [2, nil_1]
-----------------------------
removeatom    [1]
removeatom    [2]
unify         [1, 2, 1, 1]
```

図 3: 2 つ目のルールに対する命令列

に出現するアトムに対して，inheritlink 命令を用いる．この命令で使用するリンクの左辺での出現を取得するために getlink 命令を用いる．

- 右辺に 1 回出現するリンクのうち，‘=’ アトムに出現するアトムには unify 命令を用いる．

ルール中のリンクはちょうど 2 回出現する．左辺に 2 回出現するリンクはマッチングを表し，deref と eqatom 命令を使って検査できる．以上から，膜のないルールはこの命令セットでコンパイルできることが分かる．

4 アトム再利用による最適化

左辺にマッチしたアトムを再利用して右辺のアトムを生成し，命令列の最適化を行う手順を以下に示す．

1. 左辺・右辺のアトムの中から，再利用する組み合わせを決定する．
2. 再利用するアトムの除去・生成命令を取り除く．
3. 右辺のアトムを参照する変数番号を修正する．

また，この処理により明らかに冗長な getlink 命令と inheritlink 命令ができるので，これらは除去できる．

例えば，図 1 の 1 つ目のルールに対してアトム再利用を行うと，図 2 の命令列を**図 5** のように最適化できる．

```
deref        [-dstatom, srcatom, srcpos, dstpos]
findatom     [-dstatom, srcmem, func]
func         [srcatom, func]
eqatom       [atom1, atom2]
neqatom      [atom1, atom2]
removeatom   [srcatom]
newatom      [-dstatom, srcmem, func]
newlink      [atom1, pos1, atom2, pos2]
unify        [atom1, pos1, atom2, pos2]
getlink      [-link, atom, pos]
inheritlink  [atom1, pos1, link2]
removemem    [srcmem, parentmem]
newmem       [-dstmem, srcmem]
movecells    [dstmem, srcmem]
testmem      [dstmem, srcatom]
loadruleset  [dstmem, ruleset]
lockmem      [-dstmem, freelinkatom]
anymem       [-dstmem, srcmem]
eqmem        [mem1, mem2]
norules      [srcmem]
```

図 4: 命令セット

```
findatom   [1, 0, append_3]
deref      [2, 1, 0, 2]
func          [2, cons_3]
-----------------------------
getlink    [6, 1, 2]
getlink    [8, 2, 1]
newlink       [2, 1, 1, 2]
inheritlink   [2, 2, 6]
inheritlink   [1, 0, 8]
```

図 5: 図 2 を最適化した命令列

5 まとめ

LMNtal 処理系の内部命令セットおよびルールコンパイラの動作手順について説明した．次に，命令列にアトムの再利用による最適化を行う様子を紹介した．また，命令セットが，膜を含まないルールの動作を記述するために十分であることについても触れた．

今後は膜のあるルールに対する最適化を行う方法をまとめる．

参考文献

[1] 上田 和紀, 加藤 紀夫: “言語モデル LMNtal”, コンピュータソフトウェア, Vol.21 (2004) (掲載予定）.

[2] 矢島伸吾, 永田貴彦, 加藤紀夫, 上田和紀: “LMNtal プロトタイプ処理系の設計と実装”, 日本ソフトウェア科学会第 20 回記念大会論文集, pp. 21–25, 2003.

[3] Holzbaur C., Fruehwirth T., A PROLOG Constraint Handling Rules Compiler and Runtime System, *Applied Artificial Intelligence*, **14**(4), 2000, pp. 369–388