6J-08

グラフ書換え系言語 LMNtal における パターンマッチングの実行時最適化

柳川 峻広 †
早稲田大学 大学院基幹理工学研究科 †

上田 和紀 ‡
早稲田大学 理工学術院情報理工学科 ‡

1 はじめに

グラフ構造は，プログラミング言語における変数と値の関係，リスト，木などのデータ構造から，実社会における鉄道の路線や SNS のフォロー関係までさまざまな接続構造が表現できる．

LMNtal とはこのグラフ構造を直観的に扱って計算ができるグラフ書換え系言語である．LMNtal はグラフと書換えルールからなる．書換えルールは書換え前の部分グラフ，書換え後の部分グラフからなる．グラフ書換え系では全体のグラフの中にルールにマッチする部分グラフが見つかれば書換えが行われて処理が進んでいく．

LMNtal の実行時処理系 SLIM は，LMNtal コンパイラが出力する中間命令列を解釈実行する．そのため，書換えルール適用時の部分グラフを探索する命令列は静的に決まる．しかし，グラフの状態によっては静的に決定した命令列では非効率的な探索をすることがある．

そこで，グラフ全体の状態に応じて探索手順を動的に決定することで，ルール適用時のバックトラックの回数を抑える手法を提案する．

2 LMNtal とその実行時処理系 SLIM

2.1 LMNtal グラフとルール

LMNtal [1] ではグラフ理論における頂点はアトム，辺はリンクに対応する．アトムはアトム名と順序付きのリンクからなる．アトム名とリンク数の組をファンクタといい，SLIM [2] では同一のファンクタごとに連結リストでアトムのデータを管理している．書換えルールは書換え前と書換え後の部分グラフからなり，書換え前をヘッド，書換え後をボディという．ヘッドとボディにそれぞれ 1 回ずつ出現するリンクは自由リンクと呼び，書換え前のリンク先を書換え後の構造につなぎ替える．ヘッドまたはボディに 2 回出現するリンクは局所リンクと呼び，そこに書かれたアトム同士が繋がれていることを表す．

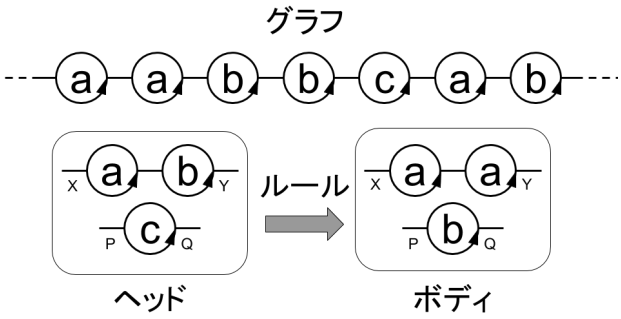

図 1 グラフと書換えルールの例

2.2 書換えルールのパターンマッチング

あるルールが適用できるかどうか検査することをパターンマッチングという．ヘッドにおける連結成分を連結パターンとする．SLIM は連結パターン内の 1 つのアトムを始点アトムとし，そこからリンクを辿るようにして検査を行う．LMNtal コンパイラは連結パターン内の始点アトムを静的に決定し，そのアトムのファンクタを引数にした findatom 命令を出力する．連結パターンが複数ある場合，その数だけ findatom 命令が出力される．グラフと書換えルールの例と，それに対応する中間命令列の一部を図 1，図 2 に示す．これは a と b が繋がっている連結パターンと c を探索し，同時に書き換えるルールである．SLIM は findatom 命令で指定されたファンクタからアトムリストを参照し，先頭から順にアトムを見ていき，それを始点アトムとしたときに連結パターンに合うかどうかを検査する．検査する中間命令に失敗した場合，直前の findatom 命令までバックトラックし，アトムリストからまた次のアトムを選択して再検査する．アトムリストの末尾のアトムの検査が失敗した場合，その findatom 命令はバックトラックを起こす．検査が全て成功したとき，書換えが行われる．

Runtime optimization of pattern matching in graph rewriting language LMNtal

† Takahiro Yanagawa, Dept. of Computer Science and Enginerring, Waseda University
‡ Kazunori Ueda, Dept. of Information and Computer Science, Waseda University

```
...
findatom      [1, 0, 'a'_2]
deref         [2, 1, 0, 1]
func            [2, 'b'_2]
findatom      [3, 0, 'c'_2]
commit          ["_XabY", 0]
...
```

図 2　中間命令列

3　問題と提案手法

3.1　findatom 命令における問題と動的にする手法

LMNtal コンパイラは中間命令を静的に出力する．すなわち，ルールのヘッド内の連結パターンの始点アトムおよび，検査命令列が静的に決定する．しかし，グラフの状態に依って静的に決定した命令列ではバックトラックの回数が多い非効率的な探索をしてしまうことがある．これを解決する方法として，グラフの全体の状態に応じてルールのパターンマッチングの探索順序を動的に決定する手法を提案する．

始点アトムの候補が複数ある，すなわちアトムを取り出すアトムリストの候補が複数あるとき，アトムリストのサイズが最小のアトムを優先的に始点アトムとすれば，検査の回数は最大でもそのアトムリストのサイズに抑えられて，バックトラックの回数を減らすことができる．

しかし，SLIM は基本的に中間命令列に従ってパターンマッチングを行うため，探索順序を動的に決定することは困難である．静的にあり得る探索順序を全列挙することも考えられるが，findatom 命令の個数の階乗に比例した中間命令列が必要になり，現実的ではない．

そこで，動的にパターンマッチングを行う実行時処理系を新たに設計する．SLIM の findatom に対応する動的な findatom のアルゴリズムの擬似コードを図 3 に示す．

3.2　動的に始点アトムを選択することの利点

LMNtal プログラムは多くの場合，ある計算フェーズを表すアクティブアトムと，データを表すデータアトムの 2 つに分けられる．アクティブアトムは常にグラフ上に 0 個もしくは 1 個であることが多い．一方でデータアトムはアクティブアトムに比べて多い．そのような状態で findatom の始点にデータアトムを選んでしまうと，バックトラックを起こしやすい．また，そもそも特定のアクティブアトムが存在しないフェーズにおいて大量のデータアトムがあるアトムリストを先に検査するのは自明に非効率的である．しかし，図 3 のアルゴリズムは小さいアトムリストを優先的に選択するため，0 個の存在

```
bool findatom(ルール,レジスタ) {
  functor <- 未決定かつ最小アトムリストのファンクタ
  if (未決定アトムがない) {
    return true
  } else {
    for (atom : functor のアトムリスト ) {
      if (atom を始点にしたときに連結パターンを満たす) {
        if (findatom(ルール, レジスタ)) {
          return true
        } else {
          レジスタから連結パターンを除去
        }
      }
    }
    return false
  }
}
```

図 3　動的な findatom の疑似コード

しないアトムに関しては即座にルールに失敗してバックトラックの回数を最小限に抑えることができる．

また，データアトムのみで構成される連結パターンに対する書換えルールがあったときを考える．異なるデータアトムの量の比が予測できないときや，大小が途中で逆転するような場合に対しても自動で対処することができる．

4　まとめ

LMNtal 実行時処理系 SLIM では，書換えルールを満たす部分グラフの探索操作が静的に決まるため，グラフの状態によってはパターンマッチング中にバックトラックが多く発生する場合があった．そこで，アトムの個数に応じて動的に探索順序を決定することにより，バックトラックの回数を抑える手法を提案し，その利点について考察した．

今後の課題として，この手法のプロトタイプ実装および SLIM との性能比較が挙げられる．動的に探索することによるオーバーヘッドを差し引いても効率化されていることが確認出来れば，新たな処理系として発展させていきたい．

参考文献

[1] 上田 和紀 and 加藤 紀夫. “言語モデル LMNtal”. In: コンピュータソフトウェア, *Vol.21, No.2* (2004), pp. 126–142. DOI: 10.11309/jssst.21.126.

[2] 村山 敬 et al. “階層グラフ書換え言語 LMNtal の処理系”. In: コンピュータソフトウェア, *Vol.25, No.2* (2008), pp. 47–77. DOI: 10.11309/jssst.25.2_47.