レイヤードストリームを用いた並列構文解析

5R-2

奥村晃、松本裕治

(財)新世代コンピュータ技術開発機構

1.はじめに

近年、GHC[Ueda 85]等のcommitted-choice language(以下、CCL)で探索問題を解く試みが幾つか報告されている。筆者等によるレイヤードストリーム・プログラミング[奥村87]もそのうちの一つで、探索プログラムを直接CCLで書き下すためのプログラミング・パラダイムを与えている。レイヤードストリーム・プログラミングの基本的な考え方は、プロセス間のデータの流れをスムースにし並列度の高い処理を行うために、先頭要素が共通であるような複数のリストを一つの構造としてまとめることである。これは、松本による並列パーザ、PAX[Matsumoto 86]で解析情報の伝達に用いられている手法を一般化し、構文解析以外の問題においても並列度を得ることが出来るようにしたものである。

レイヤードストリーム・プログラミングは探索問題一般に適用することが可能であるが、これをあらためて構文解析に用いてはどうであろうか。手法が一般化されているため、PAXほどの処理効率は望めないにしても、より高い並列性を抽出できるのではなかろうか。本報告では、こういった考えに基づいて作成したレイヤードストリーム・パーザ(以下、LSP)について紹介し、PAXとの比較を基に将来の可能性を探る。

2.並列パーザ、PAX

まず、比較対象であるPAXについて簡単に紹介する。PAXでは、解析を行う文中の終端記号とそこからボトムアップに生成される非終端記号の各々についてプロセスを発生し、その間をストリームで結合して解析情報を流す。解析情報は、ある文法規則の右辺のどこまで解析が進んだかを示す識別記号と、その規則が適用された箇所より以前の部分からのストリームとの組である。

例えば、np→det,adj,n(規則1)という規則があったとしよう。detとadjの間、及びadjとnの間にはあらかじめユニークな識別記号が割り当てられる。ここでは仮にそれぞれid1及びid2としておく。今、ボトムアップに非終端記号 detが生成されたとしよう。規則1の右辺でid1の所まで解析が出来たことになる。detに対応してプロセスが発生し、id1(X)をdetの直後のプロセスにストリーム出力する。ただしXはそのプロセスへの入力であり、文中でdetより前の部分の解析情報のストリームである。id1(X)を受け取ったのがたまたまadjのプロセスだったとしよう。id1まで解析が出来ていたところにadjが続いていたのだから、規則1のid2まで解析できたことになる。よってadjはid2(X)を出力する。さらにそれを受け取ったのが、nのプロセスだったとしよう。これで規則1の右辺が全て終了したことになるので、新たにnpのプロセスが発生する。当然ながら、det、adj、nについて他にも適用できる規則があれば、それら全てについて解析情報が発生し、それらはストリームとなって順次処理される。このようにしてボトムアップに非終端記号のプロセスを発生し、最終的に文全体から開始記号を生成することが出来れば解析は成功となる。

PAXが並列処理に有利なのは、id1(X)等の解析情報が入力Xを待たずに出力出来る点である。従来の逐次型パーザのように各部がそこまでの解析結果を受け取って出力を決定する方法とは異なり、各部が同時に処理を開始できるため高い並列性が期待できる。

3.レイヤードストリーム・プログラミング

PAXの並列度の高さははストリームを流れるデータの構造に依存している。id1(X)が示すのは、ストリームXの要素の各々に続けてid1で示される解析が行われたことである。つまり、Xの中身が$X_1, X_2, \cdots$だとすると、出力として

$$[(X_1, id1), (X_2, id1), \cdots]$$

であるところを共通するid1でまとめあげている。

これを一般化したのがレイヤードストリームである。レイヤードストリームはその要素となる構造の一部に別のレイヤードストリームを含む再帰的複合構造である。レイヤードストリームの一般形は、$[X_1{*}LS_1, X_2{*}LS_2, \cdots]$のような形をしている。$LS_i$はレイヤードストリームで、$X_i{*}LS_i$は$LS_i$の各要素と$X_i$のペアの集りを表現する構造である。例えば、1*[2*[3*begin,4*[]],5*begin]を通常のリストの集りで表現すると、{[1,2,3],[1,5]}となる。ここでbeginはレイヤードストリームの終端を示す記号である。*の後ろが空リストの場合はペアを作れないので、そのような構造は空リストを表現する。

レイヤードストリームを用いた探索プログラムを考える。PAXの場合は文法規則の右辺の左端にある記号に対応するプロセスが入力に関わらずid1(X)のような解析情報を出力できるため、各部が同時に処理を開始できるが、これは問題の持つ制約が局所的なためである。よってこの手法をそのままの形で探索問題に一般的に適用することは不可能で、一工夫必要となる。

例としてnクイーン問題を取り上げよう。nクイーンで通常用いられるアルゴリズムは、問題をn段に分けて再帰的に解くものである。i段目におい

A Parallel Parser using Layered Stream Method
Akira Okumura and Yuji Matsumoto
Institute for New Generation Computer Technology

てi-1段目までの部分解を入力とし、そこにi番目のクイーンを競合しないように置いてi段目までの部分解とするものである。これをCCLで行うためには各段のプロセスの入出力を全ての部分解によるストリームにすれば良い。入力をInとしよう。i段目のクイーンの位置として1からnまであり、それらとInとの全組み合わせは[1*In,2*In,…,n*In]となる。ここからクイーンがぶつかる場合を除いて出力すれば良い。Inの要素でkと競合するものを除いた残りをOut_kとすると出力は、

$[1*Out_1,2*Out_2,\cdots,n*Out_n]$

で表される。結局nクイーンの各段はそのリストを出力する部分と、kとInからOut_Kを求める部分から構成される(図1)。各段の出力はOut_kが求まるのを待

```
queen(In,Out) :- true |
   filter(1,In,Out1),
   filter(2,In,Out2),
      :
   filter(n,In,Outn),
   Out=[1*Out1,2*Out2,...,n*Outn]..
```

図1.nクイーン・プログラム(一部)

たずに実行できるため、並列性が高まる。

各段での可能な候補を出力するのと、その候補と競合しない残りの部分を入力から抽出するのを並行して行うのがレイヤードストリーム・プログラミングである。PAXは基本的には前段までの解析情報を基に出力を決定するのでレイヤードストリーム・プログラミングとはいえない。それではレイヤードストリームを用いたパーザ(LSP)はどんなものであろうか。

4.LSP

LSPの基本的構造を図2に示す。これは、

```
n(In,A0,B) :- true |
   A0=[np*Out1,np*Out2|A1],
   filter([det],In,Out1),
   np(Out1,A1,A2),
   filter([adj,det],In,Out2),
   np(Out2,A2,B).
```

図2.LSPプログラム(一部)

np→det,n とnp→det,adj,nに対応するものである。nクイーンの場合と違うのは、出力とfilterの他に規則の左辺の記号のプロセスの起動を行う点である。この場合はnpが2つ起動され各々直前のfilterの出力を入力とする。PAXと違うのは、PAXがdetやdet,adjが来ない限りnpを出力しないのに対し、LSPでは直ちに出力する点である。これによりプロセス間の逐次性が減少し並列度が増す。

5.PAXとLSPの比較

前節までで述べたように、レイヤードストリーム・プログラミングはPAXから基本的アイディアを得て一般化したもので、これに基づくLSPはPAXを最も並列性を示すことが可能である。ただし、そのために闇雲に出力を行うのでストリームの長さは指数的に大きくなることが確実である。そうなると、ストリームをfilterが逐次に検査するのでは並列ステップ数で不利になるが、もしストリームを一度に展開することが可能でプロセッサが無尽蔵なら、[Rytter 85]で示されたアルゴリズムと等価な処理がLSPで可能であり、処理ステップ数O(log N)で終了することができる。同じ条件の下でPAXがO(N)であることが示されているので[松本 86]、以上の仮定の下ではLSPはPAXより短い処理ステップ数を実現する。

ストリームを分解するコストが無視できる程度であるという仮定は妥当であろうが、プロセッサが無限に存在することは実際には考えられない。よって処理効率を理論的に可能な処理ステップ数で比較することはあまり意味がない。プロセッサ数、1ステップ当たりのコスト、通信時間等を考慮した比較が必要である。

LSPの出力を最も抑制した形がPAXであるといえる。PAXにおいて文法規則の部分適用が成功したものだけを出力するのは妥当であると思われるが、最適解かどうかはいまだ不明である。LSPの出力を適当に抑制することにより、現実的にもより処理効率の良いパーザが得られる可能性がある。

6.おわりに

LSPのプログラムは既に種々存在し、現在並列処理ステップ数でPAXとの比較を行っているが、現状ではPAX以上の効率を示すプログラムは作られていない。これには最適化の度合いの違いもあって、まだ正確な比較はできていない。

今後は各プログラムの最適化を進めるとともに、ストリームの展開コストを無視しプロセッサ台数等を考慮した比較方法を検討し、現実的で信頼性のある比較結果を求めていきたい。そして、それに基づいて並列パーザの最適解を得たい。

参考文献

[奥村 87] 奥村 晃,松本裕治:"レイヤドストリームを用いた並列プログラミング", Proc. Logic Programming Conference '87, Jun. 1987.

[松本 86] 松本裕治:"並列構文解析とその評価",冬のLAシンポジウム, Feb. 1986.

[Matsumoto 86] Y. Matsumoto: "A Parralel Parsing System for Natural Language Analysis", Proc. 3rd ICLP, Jul. 1986.

[Rytter 85] W. Rytter: "Parallel Time O(log N) Recognition of Unambiguous OF CFLs", Lecture Notes in Computer Science 199, Foundation of Computer Theory, GDR, Sep. 1985, pp. 380-389.

[Ueda 85] K. Ueda,: "Guarded Horn Clauses", ICOT Tech. Report TR-103.