ソフトウェア基礎論19－11
プログラミング言語 9－11
（1986. 12. 12）

並列オブジェクト指向言語による並列構文解析

A Parallel Parser in ABCL/1

大澤一郎　米澤明憲
Ichirou Ohsawa and Akinori Yonezawa
東京工業大学理学部情報科学科
Tokyo Institute of Technology

あらまし　自然言語の文を高速に解析するために、文脈自由文法の構文解析アルゴリズムを並列オブジェクト指向モデルの立場から検討し、個々の文法規則をオブジェクトとして分散させて構文解析を並列に行う並列構文解析アルゴリズムを提案する。

Abstract　A parallel parsing algorithm for context-free grammars is presented, which is designed with an object-oriented information processing model.

1. はじめに

ＧＰＳＧ[2]を中心とする最近の言語理論によれば、ほとんどの自然言語の文法が文脈自由文法の範囲で記述できる。これにともなって、自然言語を処理する分野において、文脈自由言語の文を高速に解析するアルゴリズムの必要性が高まっている。

文脈自由言語の文の構文解析アルゴリズムは、現在までに形式言語学の分野でかなり研究されており、ＣＫＹアルゴリズムやアーリーのアルゴリズムなど[5]が提案されている。これらのアルゴリズムは、一度行った計算を二度と繰り返さないという工夫により、逐次的に解析を行って、最悪の場合で入力文の長さの３乗のオーダの時間で解析を行う。

これらのアルゴリズムの解析時間はほぼ最良であるが、実際にこれらのアルゴリズムを用いて自然言語の文を解析しようとしても、機械翻訳や実時間翻訳などで必要な解析速度がなかなか得られない。そして、現在の計算機よりも基本的な演算速度の遅い人間が、計算機よりも高速に自然言語を理解している理由を考えてみると、人間は自然言語を理解する時の処理の多くを並列に行なっているためと考えられる。

本文は、文脈自由言語の構文解析アルゴリズムを並列オブジェクト指向モデルの立場から検討しなおし、アルゴリズムに並列性を導入することにより、より最適な構文解析アルゴリズムを提案しようとする予備的報告である。また、同様な研究が佐藤等[9]により進行中である。

以下では、２節で並列オブジェクト指向言語ＡＢＣＬ／１を概説し、３節で並列構文解析および４節でそのＡＢＣＬ／１による実現について述べる。５節では本方式の長所と短所を考察する。

2. 並列オブジェクト指向言語ＡＢＣＬ／１

ＡＢＣＬ／１[11]では、メッセージを受理することにより一連の動作を行うという抽象的なものを表現する「オブジェクト」の集合が、互いにメッセージのやり取りを行うことによって情報処理や計算を行う。メッセージのやり取りは系の中で複数・並列的に行うことが可能であり、複数のオブジェクトが同時に動作することができる。各オブジェクトは、それ自身の中に自立した演算能力を持ち、かつそれ自身のみが直接アクセスし、参照・更新できる局所的な記憶を保持することができる。

メッセージのやり取りには過去型と現在型と未来型の三種類の形態が用意されている。以下、これらの形態の説明において、オブジェクトＳがメッセージＭをオブジェクトＲに送ることによりメッセージのやり取りが開始されることを想定している。

過去型のメッセージＭをＲに送ったＳはさらに仕事の続きがあればそれを直ちに継続する。ＡＢＣＬ／１の構文では、この型のメッセージのやり取りを

［Ｒ　＜＝　Ｍ］

と記述する。現在型のメッセージＭをＲに送ったＳは、Ｒから何らかの返事が戻ってくるまで、残りの仕事を延期させて待機状態になる。ＡＢＣＬ／１の構文では、この型のメッセージのやり取りを

［Ｒ　＜＝＝　Ｍ］

と記述する。現在型でメッセージを送られたＲは、

！＜Ｓ式＞

というＡＢＣＬ／１の構文によって、メッセージの送り手Ｓに返事を行う。未来型のメッセージは今回のアルゴリズムでは使用しないので、ここでは省略する。

同時に複数のオブジェクトに同じメッセージを過去型で送りたい場合にはマルチキャストを用いる。ＡＢＣＬ／１の構文では、

［（Ｒ１，Ｒ２，．．．）　＜＝　Ｍ］

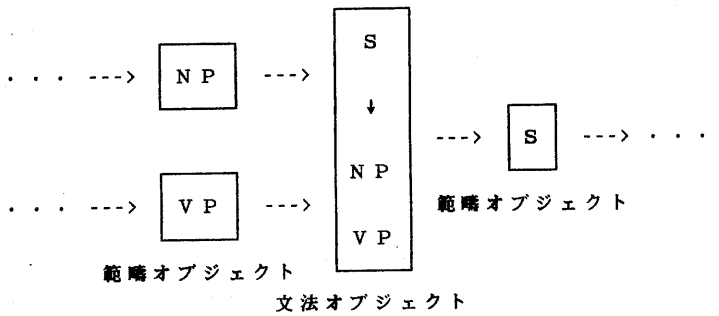
のように記述される。

3．並列構文解析

3.1.基本アイディア

構文解析アルゴリズムの並列性を高めて処理を分散させるために、本アルゴリズムでは一つ一つの文法規則をオブジェクトとして分散させる。このオブジェクトを「文法オブジェクト」と呼ぶことにする。また、各範疇に属する部分木を管理するために「範疇オブジェクト」と呼ぶオブジェクトを各範疇ごとに用意する。

図１：　Ｓ　－＞　ＮＰ　ＶＰ　を処理するモデル



この二種類のオブジェクトの基本的な挙動を簡単に説明する［図１］。文法オブジェクトは、文法の右辺にある範疇を管理している範疇オブジェクトからそれに属している完成した部分木の情報を受け取る。部分木の情報の中にはその部分木を構成する単語の並びを表わす区間（始点と終点）が含まれている。文法規則にしたがってこれらの入力データから左辺の範疇に属する部分木が完成すると、その部分木とそれを構成した区間を左辺の範疇を管理している範疇オブジェクトに送る。

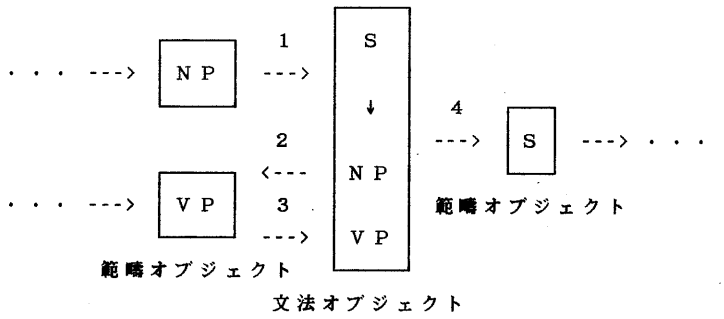
範疇オブジェクトは、文法オブジェクトから完成した部分木の情報を受け取ると、自分が左辺に属する文法を管理している文法オブジェクト全てにマルチキャストを用いてその部分木およびその区間を送る。このように本アルゴリズムは、範疇オブジェクトと文法オブジェクトをノードとする（サイクルを含む）有向グラフの中を部分木とその区間というデータが流れてボトムアップ（上向き）に解析を進めていくという単純なモデルが基本となる。

3.2.アルゴリズムの改良

上記の計算モデルでは、文法規則の右辺に属する範疇の部分木全てが独立に各文法オブジェクトに集められている。しかしながら、単語の入力が左から順番に行われることを考慮すると、文法規則の右辺最左の範疇に属する部分木が得られた時点で残りの範疇に属する部分木を管理する範疇オブジェクトに要求する方式のほうが、無駄な処理をあまりせずに済むと考えられる。さらに、必要とされている範疇以外を作らないように制御することも可能となる。

それゆえ本アルゴリズムでは、文法規則の右辺最左の範疇に属する部分木はその範疇オブジェクトから無条件に送信してもらい、文法オブジェクトは受信した部分木にしたがって残りの必要な範疇の部分木をその始点制約とともに適当な範疇オブジェクトに要求する。要求を受けた範疇オブジェクトは既に該当する部分木があれば直ちにそれを要求された文法オブジェクトに送信する。さらに要求された内容を記録しておき、後ほど要求にあてはまる部分木ができた時にはその部分木もその文法オブジェクトに送信する［図２］。

図２：　改良したモデル



1：文法規則の右辺最左範疇に属する完成した部分木の入力
2：右辺の次の範疇の部分木の要求
3：要求された部分木の送信
4：完成した部分木の送出

4．ＡＢＣＬ／１による並列構文解析アルゴリズムの実現

この節では、前節の並列構文解析アルゴリズムを並列オブジェクト指向言語ＡＢＣＬ／１により実現したのでその概要を説明する。詳しくは付録に載せた実際のプログラムを参考にして欲しい。

4.1. 範疇オブジェクト

個々の範疇に属する完成した部分木を管理するオブジェクトで、以下の三つの局所記憶と三つのメソッドを有する。

```
局所記憶
  完成木リスト                    ; 完成した部分木とその構成区間を記録する
  文法オブジェクト・リスト        ; この範疇を右辺の最左とする文法規則を表わす文法
                                  ; オブジェクトのリスト
  要求リスト                      ; 文法オブジェクトからの要求を記録する
メソッド
  [: 登録 文法オブジェクト]       ; 文法オブジェクト・リストに登録する
    文法オブジェクト・リストに文法オブジェクトを登録する
  [: 入力 部分木 :区間 始点 終点]  ; 完成した部分木の入力を受けつける
    入力された部分木を完成木リストに付け加える
    もし要求リストの中にこの部分木を要求している文法オブジェクトがあれば
        これを要求している文法オブジェクトに送信する
    文法オブジェクト・リストの中の文法オブジェクトにこの部分木を送る
  [: 要求 始点 :ＩＤ 番号]        ; 部分木の要求を受けつける
    要求リストに要求を付け加える
    もし完成木リストの中に要求を満たす部分木があれば
        その部分木を要求してきた文法オブジェクトに送信する
```

4.2. 文法オブジェクト

文法規則に従ってある範疇に属する部分木を構成するオブジェクトで、文法規則ごとに一つずつ作られる。以下に局所定数とあるのは各文法オブジェクトが生成される時に設定される。局所定数である構成規則には、右辺最左の範疇に属する部分木が入力された場合に、どのように左辺の範疇に属する部分木を構成するかが設定されている。例えば、文法規則"Ｓ －> ＮＰ ＶＰ"に対して、（ＶＰ －> Ｓ）という構成規則が設定される。これは、ＶＰという範疇に属して区間が隣接する部分木があれば、Ｓという範疇に属する部分木が構成できると解釈する。

```
局所定数
  構成規則                        ; 文法規則左辺の範疇に属する部分木の構成方法
局所記憶
  待機リスト                      ; 未完成の木を記憶する
  待機ＩＤ                        ; 待機リスト内での識別番号（初期値０）
メソッド
  [: 入力 部分木 :区間 始点 終点]  ; 右辺最左の範疇の部分木の入力を受けつける
    入力された部分木を用いて構成規則を実行するにするように自分自身にメッセージを送る
  [: 実行 規則残り :区間 始点 終点 :木 部分木リスト]
                                  ; 構成規則を実行する
    もし部分木が完成した（規則が －> に達した）ならば
```

```
        完成した部分木を管理する範疇オブジェクトに送信する
    さもなくば、もし規則の始めが範疇（残りの規則の初めがアトム）ならば
        待機ＩＤを１つ増やす
        現在の状態を待機リストに記録する
        それを管理する範疇オブジェクトに完成した部分木を要求する
    さもなくば（規則の始めは制約検査で）、制約を満たしていれば
        規則の残りを実行するように自分自身にメッセージを送る
[: 供給 ＩＤ 部分木 ：区間 終点]    ；要求した部分木の供給を受けつける
  待機リストの中からＩＤ番号にしたがって、待機していた実行途中の状態を取り出し、
  実行を再開するようにメッセージを送る
```

4.3. 解析モニター・オブジェクト

構文解析システムとユーザとの間を仲介するオブジェクトで、文法規則の登録や解析する単語列の受け付けなどを行う。

```
局所記憶
  名前表                              ；名前から範疇オブジェクトを引く表
  単語位置カウンター                  ；何番目の単語かをカウントする
メソッド
  [: 文法 左辺 ：ｉｓ 右辺 ：部分木 部分木]
                                      ；文法規則の登録
    必要な範疇オブジェクトを生成し、名前表に載せる
    文法オブジェクトを一つ生成し、規則を登録する
    右辺最左の範疇オブジェクトに今生成した文法オブジェクトを登録する
  [: 入力 単語]                       ；解析する単語の受け付け
    名前表を用いて範疇オブジェクトを見付ける
    その範疇オブジェクトにこの単語からなる部分木を入力する
```

---

5. 議論

この報告で紹介した並列構文解析アルゴリズムは、必要なメッセージの数をなるべく減らし、計算負荷が特定のオブジェクトに偏らないようにしている。例えば、文法規則を一つの場所で管理するとそこに負荷がかかるので、文法をオブジェクトにして分散させた。また、オブジェクトを生成するオブジェクトに負荷がかかるのを避けるために、解析時には一つも新しいオブジェクトを生成しないようにした。その代わりに、事前に文法にしたがって解析を行うためのネットワークを構築しておき、そのネットワークに沿って情報を流すことで解析を行う方法を採用した。

本アルゴリズムのような左隅構文解析法の場合には、左側での解析によって得られるトップダウン予測情報を用いて、無駄な処理を減らすのが普通である。ところが、その処理を並列オブジェクト指向モデルの中で行おうとするとうまくいかないことがある。佐藤[9]は、単語一つに一つの割合で目標を記録するオブジェクトを生成することでトップダウン情報を扱っているが、これらのオブジェクトにメッセージが集中する可能性がある。松本[4]は並列論理型言語を用いて、このトッ

プダウン予測情報をうまく扱っている。しかしながら、それは並列論理型言語のストリームを多用しており、その性能は現実にこのストリームがどのように実現されるかに深く依存しており、現時点ではまだ評価しづらい。

6. おわりに

並列構文解析アルゴリズムにおいて、トップダウン予測情報をどのように扱うかは非常に難しい。現在考えているのは、トップダウン予測情報を利用するせず、その代わりに双方向に必要な範疇の部分木を補っていく方式である。例えば英語において、最初の名詞句で動詞を欲するのではなく、動詞が直前の名詞句を欲するようにするのである。これはＨＰＳＧ[7]の構文解析方法として適していると考えられ、しかもここで紹介した並列構文解析アルゴリズムを若干拡張するだけで実現可能と考えられる。

謝辞

京都大学工学部長尾研究室の佐藤理史氏からは、並列構文解析に関して有益な助言をいただきました。ここに深く感謝いたします。また、ＡＢＣＬ／１の使用に関して柴山悦哉助手、佐野勝也君、本田康晃君、高田敏弘君に助力を得たので感謝します。

参考文献

[1] Feldman, J.A. and Ballard, D.H.: Connectionist Models and Their Properties, Cognitive Science 6, 205-254, 1982.

[2] Gazdar, G. and et.al.: Generalized Phrase Structure Grammar, Oxford, Basil Blackwell, 1985.

[3] Martin, C.E. and Riesbeck, C.K.: Uniform Parsing and Inferencing for Learning, AAAI-86, 257-261, 1986.

[4] 松本裕治：並列構文解析，情処研報，NL-53, 1986.

[5] 長尾真監修：日本語情報処理，電子通信学会，1984.

[6] Peterson J.L.（市川惇信，小林重信訳）：ペトリネット入門，共立出版，1984.

[7] Pollard C.: Generalized Phrase Structure Grammars, Head Grammars, and Natural Language, Ph.D. diss., Stanford Univ., 1984.

[8] Proudian D. and Pollard C.: Parsing Head-Driven Phrase Structure Grammar, Proceedings of the 23rd Annual Meeting of the ACL, 1985.

[9] 佐藤理史：パーソナル・コミュニケーション.

[10] Shibayama,E. and Yonezawa A.: ABCL/1 User's Guide.

[11] 米澤明憲，柴山悦哉他：オブジェクト指向に基づく並列情報処理モデルＡＢＣＭ／１とその記述言語ＡＢＣＬ／１，コンピュータ・ソフトウェア，3, 3, 1986.

[12] Waltz, D.L. and Ballard, D.H.: Massively Parallel Parsing, Cognitive Science 9, 1985.

付録

現段階で実現されているプログラムの一部を以下に示す。説明と若干異なっているのは、部分木としていた部分が全て部分木の意味として扱われていること、および解析結果を参照するための簡単なメソッドが組み込まれているためである。初期化操作の部分は含めていない。

```
;;-----------------------------------------------------------------------------
;; ＡＢＣＬ／１による並列構文解析器
;;       programmed by Ichirou Ohsawa on 12/28/86
;; Copyright (c) Tokyo Institute of Technology. All rights preserved.
;;-----------------------------------------------------------------------------

;;=====================================
;; 文法オブジェクト
;;=====================================
[object composer-creator
  (script
    (=> [:create Rule]
        ![object
           (state
             [waiting-id := 0]
             waiting-list                ; ((Id RP Spos Sem) ...)
           )
           (script
             (=> [:input Sem :range Spos Epos]
                 [Me <= [:execute Rule :range Spos Epos :sems (list Sem)]])
             (=> [:execute RP :range Spos Epos :sems Sems]
                 (temporary value)
                 (if (eq '-> (first RP)) then
                     (format t "~&Made a category from ~d to ~d" Spos Epos)
                     [value := (evaluate (third RP) Sems)]
                     [(second RP) <= [:input value :range Spos Epos]]
                  elseif (atom (first RP)) then
                     [waiting-id := (1+ waiting-id)]
                     (push (list waiting-id (rest RP) Spos Sems) waiting-list)
                     [(first RP) <= [:wanted Epos :id waiting-id]]
                  elseif (evaluate (first RP) Sems) then
                     [Me <= [:execute (rest RP) :range Spos Epos :sems Sems]]))
             (=> [:supply Id Sem :range Epos]
                 (temporary
                   [s := (assoc Id waiting-list)]
                 )
                 [Me <= [:execute (second s)
                          :range (third s) Epos :sems (cons Sem (fourth s))]]))
           (routine
```

```
            (evaluate (Sexp Sem)
              (temporary
                [ExSexp := (assign-sem Sexp Sem)])
              (exit-routine (eval ExSexp)))
            (assign-sem (Sexp Sem)
              ;意味値を入れる指定部分（数字）と実際の意味値を置き換える
              ・・・（省略）・・・
            ))]))]

;;=====================================
;; 範疇オブジェクト
;;=====================================
[object category-creator
  (script
    (=> [:create]
        ![object
           (state
             composed-list              ; ((Spos Epos Sem) ...)
             composer                   ; (Composer ...)
             waiting-list               ; ((Spos WaitingComposer Id) ...)
           )
           (script
             (=> [:regist-composer Comp]
                 [composer := (cons Comp composer)]
                 !t)
             (=> [:input Sem :range Spos Epos]
                 (temporary
                   [wl := waiting-list])
                 (push (list Spos Epos Sem) composed-list)
                 (while wl do
                   (if (eq Spos (first (first wl))) then
                       [(second (first wl)) <= [:supply (third (first wl)) Sem
                                                :range Epos]])
                   [wl := (rest wl)])
                 [composer <= [:input Sem :range Spos Epos]])
             (=> [:wanted Spos :id Id]
                 (temporary
                   [cl := composed-list])
                 (push (list Spos &sender Id) waiting-list)
                 (while cl do
                   (if (eq Spos (first (first cl))) then
                       [&sender <= [:supply Id (third (first cl))
                                    :range (second (first cl))]])
                   [cl := (rest cl)]))))])]
```