並列 HPSG パーザー

二宮 崇　鳥澤 健太郎　辻井 潤一
東京大学大学院 理学系研究科 情報科学専攻
{ninomi,torisawa,tsujii}@is.s.u-tokyo.ac.jp

1 はじめに

自然言語処理には、構文解析、情報抽出、知識獲得、機械翻訳など様々な処理があるが、中でももっとも基本となるパーザーの高速化が求められており、以前からその並列化の研究 [1, 2] が数多くなされている。しかしながらその並列化の研究の多くは CFG に関する研究にとどまっており、また実用に供せられるパーザーは実現されていない。

本研究は、主辞駆動句構造文法 (HPSG) [3] と呼ばれる文法枠組のための高速な並列パーザーを並列計算機上に実現することを目標とする。これにより、精巧かつ柔軟であるが解析時間という観点から実用的でないとみなされてきた HPSG を用い、音声機械翻訳システムなどで必要な real-time 処理を行うパーザーが構築できると期待される。

並列 HPSG パーザーの実現は、共有メモリ型並列計算機 Ultra Enterprise 10000 上で制約解消モジュールをライブラリとして並列オブジェクト指向言語 ABCL/f に組み込むことにより行う。本研究で実現された HPSG パーザーが、実時間、台数効果の 2 面から高効率であることは実験を通して確認する。

2 HPSG パージングアルゴリズム

本節では本研究が提案する並列 HPSG パージングアルゴリズムのベースとなる CKY ベースの逐次 HPSG パージングアルゴリズムについて説明し、その並列化の手法について述べる。

ここでは HPSG は、辞書規則の集合とルールスキーマの集合からなっているとする。辞書規則は、ある単語 w とそれに対応する素性構造で表現された辞書項目 ω の対のことであるとし、ルールスキーマは i) 構文木中の親子間を表現する素性構造からなる書換規則と ii) その書換規則を構成する素性構造間の制約[1]からなる文法規則のこととする。ここでは、説明の簡便のため制約の記述を省略し書換規則でルールスキーマを表現することにする。本稿では辞書規則を $\omega \rightarrow w$ と表記し、ルールスキーマを $\zeta \rightarrow \alpha\ \beta\ \gamma\ \ldots$ と表記する。

2.1 CKY ベースの逐次 HPSG パージングアルゴリズム

本研究で採用したパージングアルゴリズムは CKY ベースであり、まずその CKY ベースの逐次 HPSG パージングアルゴリズムについて説明する。

[1]一般に確定節プログラムと素性構造の構造共有で記述される。

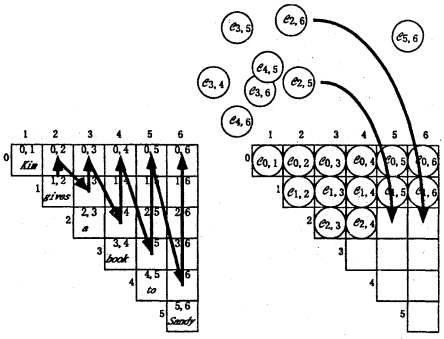

図 1: 逐次での CKY 構文解析の過程 (左) と三角行列とセルエージェントの関係 (右)

CKY ベースのアルゴリズムにおいては、ルールスキーマは $\gamma \rightarrow \alpha\ \beta$(ただし、$\gamma, \alpha, \beta$ は素性構造) という形をしており、書換規則の右側には二つの素性構造しかないものと仮定する。ここで、与えられる文を $w_1 w_2 \cdots w_n$ とし、$S_{i,j}$ を次のように定義する。

$S_{i-1,i}$: 辞書規則中で $\omega \rightarrow w_i$ を満たす全ての素性構造 ω の集合

$S_{i,j}$: 全ての $k (i < k < j)$、ルールスキーマ $\gamma \rightarrow \alpha\ \beta$、$\varphi \in S_{i,k}, \psi \in S_{k,j}$ に対し、α と φ、β と ψ を単一化し得られる全ての素性構造 γ の集合

$S_{i,j}$ を三角行列の要素とみなし、図 1 (左) の順で $S_{i,j}$ の要素を求める計算を行うと、上記の条件を満たす全ての $S_{i,j}$ が求まり、$S_{0,n}$ が求まれば構文解析は終了する。直観的には $S_{i,j}$ は $w_{i+1}, \cdots, w_j$ を葉とする部分構文木の根を表す素性構造の集合に対応し、三角行列は三角行列の右上を頂点とする構文木全体に対応する。

逐次実行の場合のアルゴリズムをプログラムで表現すると図 2 のようになる。図中の手続き "*rule-schema*(φ, ψ)" は、全てのルールスキーマ $\gamma \rightarrow \alpha\ \beta$ に対し、α, φ と β, ψ とを単一化し、単一化に成功する場合の全ての γ を値として返す関数である。

2.2 並列 HPSG パージングアルゴリズム

本研究の提案する CKY ベースの並列 HPSG パージングアルゴリズムはエージェントベースのアルゴリズ

```
procedure HPSG構文解析 ()
  forall 1 ≤ j ≤ n do
    S_{j-1,j} := {ω|ω → w_j};
  endfor

  for j := 2 to n do ..... ループA
    for i := j − 2 to 0 do ..... ループB
      for k := i + 1 to j − 1 do ... ループC
        foreach φ ∈ S_{i,k} do ..... ループD
          foreach ψ ∈ S_{k,j} do .... ループE
            W := rule-schema(φ, ψ);
            S_{i,j} := S_{i,j} ∪ W;
          foreachend
        foreachend
      forend
    forend
  forend
```

図 2: 逐次 CKY アルゴリズムのプログラム表現

ムであり、各エージェントは並列に動作し、メッセージを送受信しながら処理を進めて構文解析の計算を行う。本研究の提案するアルゴリズムは 3 つのタイプのエージェント、パージングエージェント $\mathcal{PARSER}$、セルエージェント $\mathcal{C}_{i,j}$、制約解消エージェント $\mathcal{CSA}$ からなる。

パージングエージェント $\mathcal{PARSER}$: パージングは $\mathcal{PARSER}$ に対し、文 $w_1 w_2 \cdots w_n$ を送信することにより開始される。$\mathcal{PARSER}$ はセルエージェント $\mathcal{C}_{i,j}(0 \leq i < j \leq n)$ を生成し、それらを三角行列上の要素 $S_{i,j}$ と対応させる (図 1 (右))。各 $\mathcal{C}_{i,j}$ は並列に $S_{i,j}$ を計算する。

セルエージェント $\mathcal{C}_{i,j}(j-i=1)$: $\mathcal{C}_{i,j}(j-i=1)$ は辞書規則を調べ単語 w_j に対する素性構造の辞書項目を手に入れる。

セルエージェント $\mathcal{C}_{i,j}(j-i>1)$: $\mathcal{C}_{i,j}(j-i>1)$ は以下のステップで $S_{i,j}$ の計算を行う。

1. $S_{i,k}, S_{k,j}(i < k < j)$ を含んだメッセージが $\mathcal{C}_{i,k}, \mathcal{C}_{k,j}(i < k < j)$ から届くのを待つ。
2. ある k に対し $S_{i,k}, S_{k,j}$ のペアが届いたならば、ただちに $S_{i,k}, S_{k,j}$ の要素から $S_{i,j}$ の要素を計算する。ただし、実際の計算は $\mathcal{C}_{i,j}$ により行なわれるのではなく、$\mathcal{CSA}$ によって行われる。$\mathcal{C}_{i,j}$ は全ての $\langle \varphi, \psi \rangle (\varphi \in S_{i,k}, \psi \in S_{k,j})$ の組合せを作り、各々の組合せをそれぞれ異なる $\mathcal{CSA}(\mathcal{CSA}$ は複数存在する) に送り、$\mathcal{CSA}$ により並列に計算が行なわれる。$\mathcal{CSA}$ による実行結果は $\mathcal{C}_{i,j}$ が保持する $S_{i,j}$ にマージされる。
3. 全ての $k(i < k < j)$ に対し計算が終了したならば、$S_{i,j}$ の全ての要素が求まったことになり、$\mathcal{C}_{i,j}$ は $\mathcal{C}_{l,j}, \mathcal{C}_{i,m}(0 \leq l < i, j < m \leq n)(\mathcal{C}_{l,j}, \mathcal{C}_{i,m}$ は $S_{i,j}$ の送信を待っている) に対し $S_{i,j}$ を送信し、$\mathcal{C}_{i,j}$ の仕事は終了する。

制約解消エージェント $\mathcal{CSA}$: $\mathcal{CSA}$ は各プロセッサ毎に割り当てられる。$\mathcal{CSA}$ は $\mathcal{C}_{i,j}$ から渡された素性構造 $\langle \varphi, \psi \rangle$ を部分構文木の根とみなし、$\langle \varphi, \psi \rangle$ から生成されうるより大きな部分構文木を全て求める。

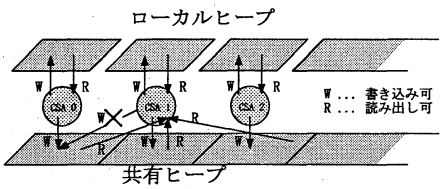

図 3: CSA のアーキテクチャ

図 2 の逐次アルゴリズムと比較すると、各 $\mathcal{C}_{i,j}$ が並列に動作することは、図中のループ A とループ B を並列実行させていることに相当する。ただし、この際、外側から 3 番目のループ、ループ C の計算は並列に実行させないが、ループ C 中で実行に必要なペア $\langle S_{i,k} S_{k,j} \rangle$ はペアが揃い次第、そのペアを用いて $S_{i,j}$ の一部を計算しているといえる。また、複数の $\mathcal{CSA}$ が並列に動作することはループ D とループ E を並列実行させていることに相当する。

$\mathcal{CSA}$ が行う単一化操作は本質的に破壊的操作であるため、副作用を伴わない正常な操作を並列環境下で実行するアーキテクチャが必要となる。次小節ではその $\mathcal{CSA}$ を実現するアーキテクチャとアルゴリズムについて説明する。

2.3 制約解消エージェント

$\mathcal{CSA}$ の実行メカニズムにおいて重要なことは、単一化操作そのものは本質的に破壊的操作であるため、複数の $\mathcal{CSA}$ が同一の素性構造に対して単一化操作を行う場合に正常に動作するメカニズムが必要であるということである (例えば、素性構造操作に対するロックなど)。本研究では、素性構造への操作の際には必ず素性構造をコピーするというアプローチをとることによりこの問題を解決する。

$\mathcal{CSA}$ は素性構造付き論理型プログラミング言語 LiLFeS [4] で用いられている抽象機械 LiAM の機能がカプセル化された制約解消のためのエージェントである。

$\mathcal{CSA}$ のアーキテクチャは LiAM の拡張であり、共有ヒープと呼ばれる記憶領域とローカルヒープと呼ばれる記憶領域を保持する (図 3)。ローカルヒープは各 $\mathcal{CSA}$ 毎に用意された素性構造の制約解消計算のための一時的記憶領域であり、各 $\mathcal{CSA}$ は他の $\mathcal{CSA}$ のローカルヒープに対して、書き込み、および、読み出しを行うことはできない。一方、共有ヒープは $\mathcal{CSA}$ 間で素性構造を交換するための領域であり、共有メモリ上で実装されることを前提とし、各 $\mathcal{CSA}$ に書き込み可能領域が割り当てられている。

$\mathcal{CSA}$ は $\mathcal{C}_{i,j}$ から、素性構造 $\langle \varphi, \psi \rangle$ の ID を含んだメッセージを受け取ると以下のステップで $\langle \varphi, \psi \rangle$ から生成される部分構文木の根を求める計算を行う。

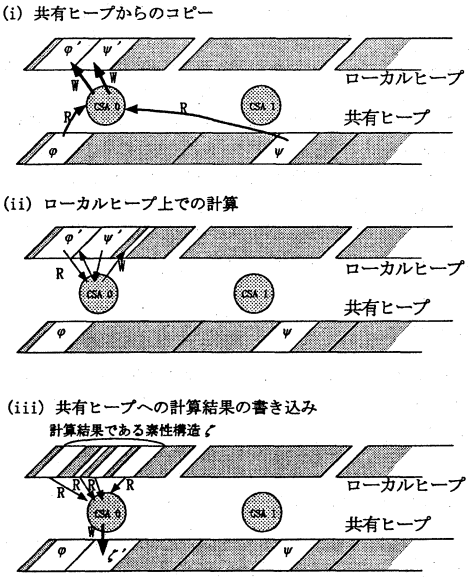

図 4: $\mathcal{CSA}$ の計算ステップ

1. φ、ψ を共有ヒープからローカルヒープにコピーをする。(図 4 (i))
2. ローカルヒープ上で φ、ψ とルールスキーマとの単一化を行う。(図 4 (ii))
3. 単一化に成功したならば、生成された部分構文木の根である素性構造を共有ヒープの書き込み可能領域に書き込み、その素性構造の ID を保持する。全てのルールスキーマに対し単一化操作を終えたら、ステップ 5 へ進む。(図 4 (iii))
4. 別のルールスキーマを選択して、ステップ 2 へ進む。
5. メッセージを送信してきた $\mathcal{C}_{i,j}$ に対し、その計算結果である素性構造の ID を全て返す。

以上で述べたアーキテクチャ、アルゴリズムにおいては次のような効率面での長所があげられる。

ロックが不要 共有ヒープは共有メモリ上に実装されるため、複数のプロセッサから同時に読み出されることが可能であるということ、および、各 $\mathcal{CSA}$ 毎に書き込み可能な領域が割り当てられているということから、ロックをすることなく独立に読み出し、書き込みを行うことができる。また、素性構造の ID は素性構造の書き込みが終って初めて他のエージェントに公開されるため、書き込まれている最中の素性構造が読み出されることはない。

台数	1	10	20	30	40	50
(a)	1,057	248	138	106	93	85
(b)	1,542					

(a) ... CKY ベースの並列 HPSG パーザー
(b) ... LiLFeS 上の CKY ベースの HPSG パーザー

表 1: CKY ベースの並列 HPSG パーザーにおける一文あたりの平均解析時間 (msec)

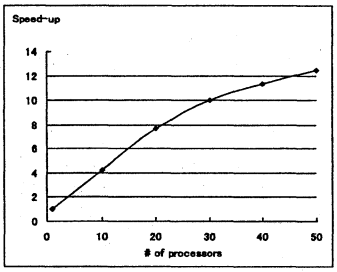

図 5: CKY ベースの並列 HPSG パーザーの台数効果

高速なコピー 共有ヒープ上の素性構造は常にヒープ上の連続した領域に存在しており、ヒープ上で断片化された素性構造のコピーに比べ非常に高速にコピーされるという特徴を持つ。また、コピーは共有メモリ上で行なわれるため、メッセージに埋め込む、送受信するといったオーバーヘッドも生じない。

要求駆動のコピー $\mathcal{C}_{i,j}$ 間では素性構造の ID のみが受渡しされ、実際に素性構造がコピーされるのは $\mathcal{CSA}$ に対し、その ID が示す素性構造への操作要求のメッセージが送られた時である。これにより、メッセージの通信による無駄なコピーを削減できる。

3 性能評価

今回 Sun Ultra Enterprise 10000 上に実現した並列 HPSG パーザーの性能評価の実験を行った。実装は並列オブジェクト指向言語 ABCL/f を用いて行い、$\mathcal{CSA}$ は LiAM をモジュールとして ABCL/f に組み込むことにより実現した。実験は、我々のグループで開発された HPSG による日本語文法 (6 つのルールスキーマ、80 の辞書項目からなる)、および、EDR コーパスからランダムに抜きだした日本語テキスト 879 文 (平均文長 20.8) を用いて行った。また、実験環境である Ultra Enterprise 10000 は UltraSparc 250 MHz 64 台からなり、6 GB の共有メモリを有する。実験の対象となったパーザーは ABCL/f で記述された並列 HPSG パーザーとその $\mathcal{CSA}$ がベースとしている LiLFeS 言語上で製作された CKY ベースの逐次 HPSG パーザーである。

実験結果は表 1 が示しており、その台数効果をグラ

台数	平均解析時間 (msec)		
	(a)	(b)	(c)
1	19,307	30,867	389,370
10	3,208		
20	2,139		
30	1,776		
40	1,841		
50	1,902		

(a) ... チャートベースの並列 HPSG パーザー
(b) ... LiLFeS 上のチャートベースの HPSG パーザー
(c) ... ALE システム

表 2: ALE スタイルの文法のためのチャートベースの HPSG パーザーにおける一文あたりの構文解析時間 (msec)

フにすると、図 5 のようになる。その台数効果は最大で 12.4 倍に達し、その一文あたりの平均解析時間は 85 msec であった。この結果から、実時間、台数効果の 2 面から非常に高効率な並列 HPSG パーザーを実現したと言える。また、1 台だけ用いた実験においては、LiLFeS 上で開発されたパーザーよりも高速であるが、この結果から並列化によるオーバーヘッド、特にコピーによるオーバーヘッドは非常に小さかったということがいえる。LiLFeS より高速であることの理由としてはベースとしている言語 ABCL/f が LiLFeS よりも高速であるということが考えられる。

4 応用

CSA は CKY ベースの HPSG パーザーだけではなく、他のアルゴリズムに基づく HPSG パーザーや、NLP システムにも流用することが可能である。本研究では、ALE システム [5] で用いられている ALE スタイルの HPSG 文法を受理するチャートパーザーの並列化も行っている。その並列化はデータの流れ、処理の並列性という観点からは、CFG のための並列チャートパーザーである PAX[6] と同じ並列化を行っているといえる。実験は平均語長 16.7 の 3 文、および ALE システムに附属するサンプル HPSG 文法を用いて、チャートベースの並列 HPSG パーザー、LiLFeS 上のチャートベースの逐次 HPSG パーザー、および ALE システムと比較することにより行った。実験結果は表 2 に示されており、最大で 10.9 倍の台数効果が得られている。この結果より、1 台の場合で比較しても LiLFeS 上のパーザー、および、ALE システムと比較しても高速であること、また高い台数効果が得られていることがわかる。

5 結論と将来の課題

実時間、台数効果の 2 面から高効率な並列 HPSG パーザーを実現した。本稿における実験、および考察を通して、本研究のアルゴリズムが非常に有用であること、および、素性構造のための並列計算モジュールが非常に高効率で汎用性をもつことを示し、また、現実に流通するテキストを用いることによって、本研究で提案する並列 HPSG パーザーが実用に耐えうることを示した。

制約解消のベースとなっている LiLFeS は現在 WAM ベースのエミュレーターであるが、ネイティブコードへの変換を含む LiLFeS の最適化手法 [7]、選言的素性構造による高速化の手法 [8] を用いることにより、さらに高速化されることが期待される。将来的には、HPSG・DRT に基づく照応、時制の解析 [9] の手法を組み込むことにより、並列 HPSG パーザーを核とする高速かつ精度の高い対話システムが構築されることが期待される。

参考文献

[1] Adriaens and Hahn, editors. *Parallel Natural Language Processing.* Ablex Publishing Corporation, New Jersey, 1994.

[2] Andrew Haas. Parallel parsing for unification grammars. In *IJCAI '87*, pages 615–618, 1987.

[3] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press and CSLI Publications, 1994.

[4] Takaki MAKINO, Kentaro TORISAWA, and Jun'ichi TSUJII. LiLFeS — practical unification-based programming system for typed feature structures. In *NLPRS'97*, pages 239–244, 1997.

[5] Bob Carpenter and Gerald Penn. ALE 2.0 user's guide. Technical report, Carnegie Mellon University Laboratory for Computational Linguistics, Pittsburgh, PA, 1994.

[6] Yuji Matsumoto. A parallel parsing system for natural language analysis. In *Proceedings of 3rd International Conference on Logic Programming*, pages 396–409, 1987.

[7] 吉田稔, 牧野貴樹, 鳥澤健太郎, and 辻井潤一. 素性構造処理言語 LiLFeS の最適化技術. In 言語処理学会第 4 回年次大会発表論文集, March 1998.

[8] 宮尾祐介, 鳥澤健太郎, 建石由佳, and 辻井潤一. 実用的な hpsg 文法のための二つの手法: 型の combining と選言的素性構造の packing. In 言語処理学会第 4 回年次大会発表論文集, March 1998.

[9] 金山 博, 緒方 典裕, and 辻井潤一. 照応・時制の underspecified な談話表示構造. In 言語処理学会第 4 回年次大会発表論文集, March 1998.