素性構造抽象機械と部分的単一化の実現

牧野 貴樹　西田 健二　鳥澤 健太郎　辻井 潤一
東京大学理学部情報科学科
{mak,nishiken,torisawa,tsujii}@is.s.u-tokyo.ac.jp

1 はじめに

HPSG[1] は自然言語における構文と意味を記述する枠組であり、高い記述力と柔軟性によって注目を集めているが、HPSG のパージングは効率の点で問題があった。そこで、HPSG のパージングにおける効率を主眼におき、鳥澤らが HPSG のためのパージングアルゴリズム [2] を開発した。

本研究の目標は、Carpenter によって提案された素性構造を効率的に取り扱うための抽象機械 [3] を実装および拡張し、鳥澤のアルゴリズムで用いられる部分的単一化と効率的なパージングのための素性構造のコピー機能を実現することである。

2 背景

本節では、鳥澤のアルゴリズムの概要を、本研究の対象となった部分的単一化に焦点をあてて説明をする。また、素性構造抽象機械の概要についても述べる。

2.1 鳥澤のアルゴリズムの概要

一般のボトムアップ HPSG パーザーでは、パージングの際に構文木の子の素性構造とプリンシプルあるいはルールスキーマと呼ばれる素性構造を単一化することで、構文木の親の素性構造を作成していく。だが、この手法では高コストな操作である単一化を多用することになり、非効率である。

これに対して鳥澤のアルゴリズムでは、効率を改善するためにパーズを 2 フェーズに分けて計算している。

1. 辞書項目から構文木の骨格の一部となる素性構造をパーズに先立ち計算しておく。
2. フェーズ 1 では、1. で計算された素性構造を組み合わせることによって、図 1 の囲みの中に示されるような構文木の主要部分である素性構造を計算する。
3. フェーズ 2 では、フェーズ 1 で計算された構文木の主要部分である素性構造に対して、図 1 の囲みの外に示されるような構文木の主要部分でない素性構造を補完する。補完された素性構造は構造共有を通して構文木の親の素性構造に伝播されていき、パーズを完成させる。

フェーズ 2 での補完には通常の単一化の使用も可能であるが、図 2 に示されるように構文木の親の素性構造の更新されていない部分まで生成してしまうので非効率である。そこで、鳥澤のアルゴリズムでは部分的単一化という特殊な単一化を用いている。部分的単一化は図 2 に示されているように、構文木の親の素性構造において更新された素性構造のみを生成するアルゴリズムである[1]。

2.2 素性構造抽象機械の概要

素性構造抽象機械 [3] は Carpenter らによって提案された型付き素性構造を効率的に扱う抽象機械で Warren's Abstract Machine[4] を拡張したものである。

[1]構文木の子に対しては素性構造の単一化可能性のチェックは行う。

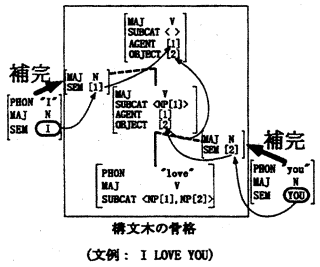

図 1: フェーズ 2 の概観

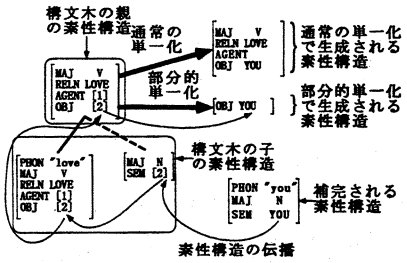

図 2: 通常の単一化と部分的単一化

2.2.1 素性構造抽象機械

素性構造抽象機械を構成する記憶領域は、以下のように構成されている。

- 素性構造の型と型階層の記憶領域
- 素性構造を格納するためのヒープ[2]
- 一時的記憶として用いられるスタック
- 単一化されるヒープ上の素性構造を指し示すためのレジスタ
- プログラムの記憶領域
- 制御変数

素性構造抽象機械は図 3 で示されるように上に述べた記憶領域とその制御部から成立しているといえる。以下に各記憶領域の説明をする。

[2]我々の実装では素性構造のコピーのために、コピーされた素性構造を記録するためのヒープが別に存在する。

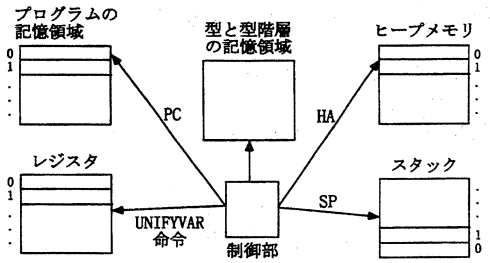

図 3: 素性構造抽象機械の概略図

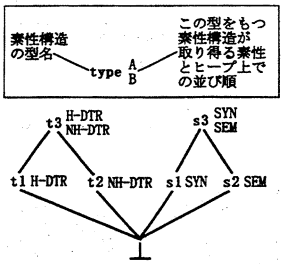

図 4: 素性構造の型階層の例

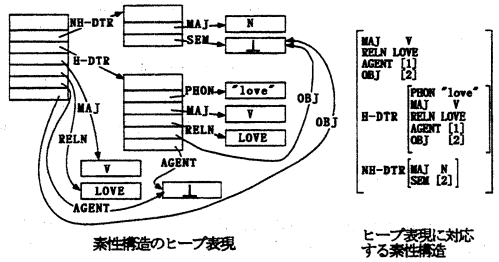

図 5: 抽象機械での素性構造の表現

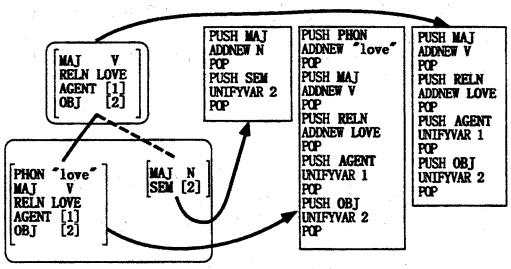

図 6: 単一化を行う抽象機械命令列への変換例

素性構造の型と型階層の記憶領域

素性構造抽象機械では型付き素性構造を使用するが、この枠組では素性構造の型に階層が存在する。この記憶領域には、素性構造の型と型階層、および素性構造が取る素性とヒープ上での素性の並び順が記録されている。よって、素性構造の型に対して取る素性が決められているので、素性構造の型が決定すれば、その素性構造が取り得る素性が決定する [3]。例えば、図 4 では素性構造が型 **t3** であれば素性 H-DTR, NH-DTR を取り得るが、他の素性を取ることはない。

ヒープ

ヒープは記憶の基本単位であるセルの配列である。ヒープ上では素性構造は、その型が取る素性の数を n とすると、$n+1$ 個の連続するセルに格納されている。その中の先頭番地のセルには、その素性構造の型名が格納されている。続く n 個のセルには、型が取る素性に対応する素性構造が記述されている。セルの順序と素性の対応は型によって決定されるのでヒープ上には素性の名前を格納しておく必要がなく、その結果ヒープ上の素性構造はコンパクトに表現される。素性構造をヒープ上の表現でで記述すると図 5 のようになる。

スタック

スタックは抽象機械命令の実行の際に、ヒープ上の素性構造へのポインタなどの一時的記憶として用いられる。

レジスタ

レジスタは、ヒープ上の素性構造の単一化を行なう抽象機械命令の実行の際に補助記憶として用いられる。

プログラムの記憶領域

抽象機械命令列で記述された素性構造を取り扱うためのプログラムが格納されている。このプログラムによって、素性構造抽象機械は素性構造に対して種々の操作を行なうことができる。

制御変数

プログラムカウンタ (PC)、次に実行される抽象機械命令が操作する素性構造が格納されているヒープ上のセルのアドレス (HA) などがある。

2.2.2 素性構造の単一化

素性構造抽象機械において、素性構造を単一化する方法としては次の 2 つがある。

1. ヒープ上の素性構造どうしを単一化する。
2. 片方の素性構造を、単一化を行なう抽象機械命令からなるプログラムに変換し、そのプログラムを、もう一方の素性構造に対して実行することにより、両方の素性構造の単一化をする。

2. の方法での単一化は、1. の方法での単一化に比べて効率的に行なうことができる[3]。本節の残りでは、単一化を行なう抽象機械プログラムで使用される抽象機械命令の動作の説明を行なう。

- PUSH 命令は、素性を引数として取り、HA が示す素性構造の素性をたどっていく命令である。実際に

[3] 2. の方法の単一化においてもその過程で 1. の方法の単一化を用いる場合があり、UNIFYVAR 命令でヒープ上の素性構造同士を単一化する場合がそれに当たる。

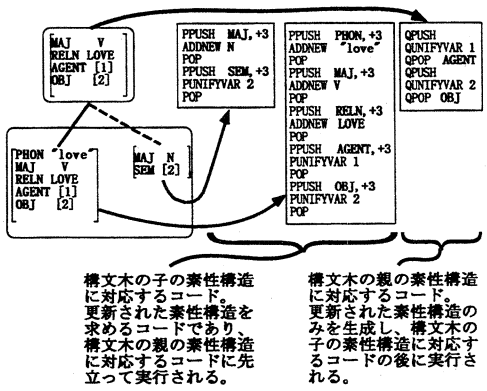

図 7: 部分的単一化を行う抽象機械命令列への変換例

は、HA の値をスタックに積み、HA の値にたどった先の素性構造の番地を代入する。もし、HA が示す素性構造に、指定された素性が存在しなければ、素性構造の型を変換してからたどる。そのとき、型が変換できなければ、素性構造抽象機械はプログラムの実行を停止する。

- POP 命令は、引数を取らず、PUSH 命令でたどった素性を逆にたどる命令である。実際には、スタックをポップして HA に代入するだけである。
- ADDNEW 命令は、素性構造の型を引数に取り、HA で示される素性構造の型と型の単一化を行ない[4] HA で示される素性構造の型を単一化された型に変更する。また、型の単一化ができなかった場合は、素性構造抽象機械はプログラムの実行を停止する。
- UNIFYVAR 命令は、レジスタを引数に取り、レジスタで示される素性構造と HA で示される素性構造を単一化する。もしレジスタが素性構造を示していなければ、レジスタに HA の値を代入する。もし単一化ができなければ、素性構造抽象機械はプログラムの実行を停止する。

単一化を行なう抽象機械プログラムへの変換例を図 6 に示す。

3 素性構造抽象機械の拡張

3.1 部分的単一化の説明

本研究では、部分的単一化を素性構造抽象機械上で実現する方法を提案する。これは、基本的には、通常の単一化と同様に、片方の素性構造を抽象機械命令列に変換し、その命令列の実行によって部分的単一化を行うというものである。ここで、部分的単一化では、構文木の子にあたる素性構造と、親にあたる素性構造で役割が異なることに対応して、本研究ではこの両者に対して別個の命令群を用意した。

- 構文木の子の素性構造のための、構文木の親に伝播されるべき素性構造[5]を検出する命令群

[4]例えば、図 4 における型 t1 である素性構造に対して ADDNEW t2 命令を実行すると、素性構造の型は t1, t2 両者の制約を満たす型 t3 になる。なお、素性構造の型の詳細については [5] を参照のこと。

[5]親に伝播されるべき素性構造とは、より上位の構文の解析で利用されるような情報が含まれている素性構造のことを指す。具体的には、ADDNEW 命令で型が変更された素性構造と、UNIFYVAR 命令で構造共有が発生した素性構造のことである。

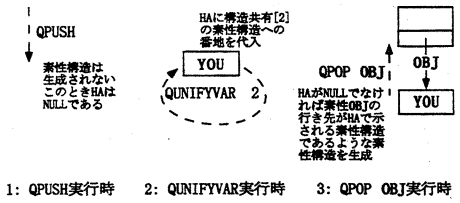

図 8: 伝播されるべき素性構造の生成

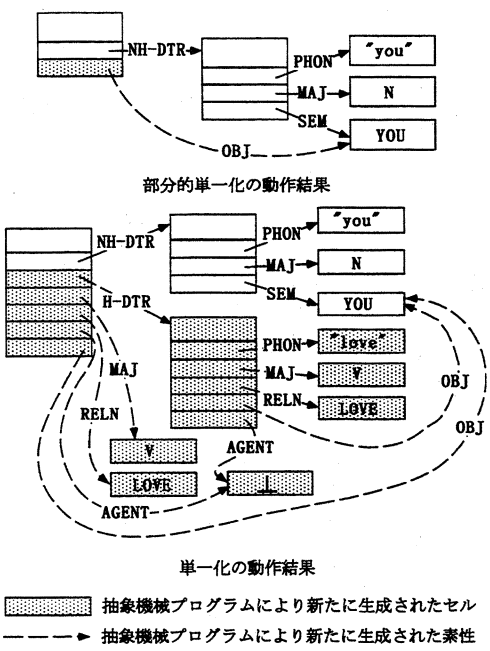

図 9: 部分的単一化と通常の単一化の動作結果の比較

- 構文木の親の素性構造のための、構文木の子の素性構造で伝播されるべき素性構造が検出された場合にのみその素性構造を生成する命令群

図 7 は、これらの命令群を使用した素性構造の変換例である。

以下の 2 節では、構文木の子の素性構造と構文木の親の素性構造の抽象機械プログラムへの変換方法について説明する。

3.2 構文木の子の素性構造の抽象機械命令列への変換

構文木の子の素性構造は、基本的には単一化と同様に操作対象の素性構造をたどる抽象機械プログラムに変換される。この部分の通常の単一化との相違点は次の 2 点である。

1. 操作対象の素性構造にたどるべき素性がない場合はたどらない。
2. 構文木の親の素性構造に伝播されるような素性構造をチェックする。

1. に対応して PPUSH 命令を、2. に対応して PUNIFYVAR 命令を導入した。

- PPUSH 命令は、素性とプログラム領域の番地を引数にとる。動作は、基本的には PUSH 命令と同じだが、HA で示される素性構造に対応する素性が存在しなければ、引数で指定された番地まで、命令を実行せず読み飛ばす。

- PUNIFYVAR 命令は、UNIFYVAR 命令の動作を行なった後、引数のレジスタ番号を HA で示されるセルに記録しておく。既に他のレジスタ番号がセルに記録されている場合は、このセルを構文木の親に伝播することにする。また、HA で示される素性構造の素性をたどっていき、たどった先のそれぞれの素性構造を表すセルに引数のレジスタ番号を記録していく。

また、プログラム中で ADDNEW 命令で型が更新された場合も、そのセルを親に伝播する素性構造として記録する。

3.3 構文木の親の素性構造の抽象機械命令列への変換

構文木の親の素性構造に対しては、前節の処理が終った後で、PUNIFYVAR 命令で伝播されるべき素性構造であるとみなされた素性構造だけを生成する。ここで、通常の単一化のように、PUSH が実行される時点で素性を生成する処理を行なうと、その素性の中に伝播される素性構造が存在しなかった場合に無駄な素性が生成されてしまう。そこで、本研究では、伝播される素性構造が存在することが確認できる時点、すなわち POP が実行される時点で素性を生成する方針をとった。

構文木の親の素性構造に対応して導入される抽象機械命令は次の 3 種類である。

- QPUSH 命令は引数を取らず、HA の値をスタックに積み HA に NULL[6] を代入する。

- QUNIFYVAR 命令は、引数で指定されたレジスタが指し示す素性構造が、PUNIFYVAR 命令で親に伝播されるとされている場合のみ、HA に引数で指定されたレジスタの値を代入する。

- QPOP 命令は素性を引数に取り、HA が NULL でないときにスタックトップが指し示す素性構造から HA が指し示す素性構造への素性を生成する。その後に、スタックをポップして HA に代入する。

図 8 は、これらの抽象機械命令の実行の様子を示す。QPUSH 命令は実際の素性構造をつくらず、QUNIFYVAR 命令で素性構造が伝播されている場合のみ、QPOP 命令で実際の素性構造をつくっていく。

3.4 部分的単一化と単一化の動作比較

図 9 に図 2 の素性構造抽象機械上での単一化と部分的単一化の動作結果を示す。単一化では、17 個のセルが新たに生成されるのに対し、部分的単一化では、わずか 1 個のセルしか新たに生成されないことが分かる。

4 実験

本研究で実装した抽象機械上での部分的単一化の有効性を示すため、単一化と部分的単一化をそれぞれ 10000 回行なって、その速度を比較した。

実験は C++ 上に実現された素性構造抽象機械を用い、DEC ALPHA 500(400MHz, 256MB RAM) 上で行われた。使用した素性構造の大きさは、抽象機械命令列になる素性構造は A,B,C 共に同じで 88 ノード、操作される素性構造は A が 55 ノード、B が 128 ノード、C が 194 ノードである。実行において生成された素性構造は、単一化においては、A,B,C の場合でそれぞれ 58 ノード、38 ノード、13 ノードであり、部分的単一化においては A,B,C の場合全てで 0 ノードであった。

	単一化	部分的単一化
A	477	235
A+ コピー	551	235
B	501	303
B+ コピー	561	315
C	869	836
C+ コピー	930	848

表 1: 実験結果 (単位: msec)

表 1に実験結果を示す。ここでコピーとは、生成された構文木の親の素性構造をコピーすることを指す。また、実験で使用される素性構造について、A,B は実際のパーズ時に用いられる素性構造であるが、C は実験用に作成した素性構造である。部分的単一化の実行時間がコピーの影響を受けないのは、構文木の親の素性構造が生成されないためである。部分的単一化は操作される素性構造が、抽象機械命令列になる素性構造に比べ小さいとき、すなわち PPUSH 命令が素性をたどらない場合に高速である。なお実際のパーズにおいては、構文木の親の素性構造はコピーされるので、コピーを含む実験結果の方が現実に近いといえる。

5 まとめ

本論文では素性構造抽象機械 [3] を実装および拡張し、高効率な HPSG パーザーに必要な部分的単一化を抽象機械上で実現した。また、論文中では特に言及しなかったが、本抽象機械上では素性構造のコピー機能も実装されている。さらに抽象機械上に Prolog ライクな素性構造をサポートする論理型言語 LiLFeS も実現されている。以上の特長により、高効率かつ柔軟な HPSG パーザーが容易に実現されると期待される。

参考文献

[1] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, 1993.

[2] Kentaro Torisawa and Jun'ichi Tsujii. Computing phrasal-signs in HPSG prior to parsing. In *COLING 96*, pages 949–955, 1996.

[3] Bob Carpenter and Yan Qu. An abstract machine for attribute-value logics. In *IWPT 95*, pages 59–70, 1995.

[4] Hassan Aït-Kaci. *Warren's Abstract Machine, A Tutorial Reconstruction*. The MIT Press, 1991. ISBN 0-262-01123-9 (hc.) — ISBN 0-262-51058-8 (pbk.).

[5] Bob Carpenter. *The Logic of Typed Feature Structures*. Cambrige University Press, 1992. ISBN 0-521-41932-8.

[6] NULL は、指し示す素性構造がないことを示す。