オブジェクト指向パーザ POWER

高橋博之　　宮崎正弘

新潟大学大学院工学研究科

1　はじめに

POWER はオブジェクト指向の考え方を用いた、従来の方式とは大きく異なるパーザである。POWER は従来のパーザの抱える記述性、拡張性の問題の改善を第一の目標とし、オブジェクト指向の考え方を用いた手続き中心の記述を行なうことでこれを実現した。

POWER は自律的に働くオブジェクト（=単語）の相互メッセージ通信により解析を行なう。オブジェクトの記述は手続きで行なわれ、オブジェクト指向の差分プログラミングの考え方により、一般的な動作をするオブジェクトに例外的動作を差分の形で加えて新しいオブジェクトを生成することができる。このため記述の重複が避けられ、また例外が分離されて見通しがよくなっている。

2　従来のパーザの問題点

従来のパーザでは単語は単なる記号であるとして、その並びに外側から補強 CFG で記述した規則を適用していた。この方法には以下のような問題点がある [1]。これらの問題点の多くは文法規則の記述のしやすさに関するものである。

1. 例外的な記述を積み重ねるとルールの数が爆発的に増えてしまい、全体の見通しが悪くなる。
2. ルールの適用順序を十分に制御できない。
3. 並列構造、係り受けの交叉をうまく記述できない。
4. 多義の爆発的増加が起こりやすい。また、何が原因で多義が発生しているのかわかりにくい。
5. 処理のモジュール化がされていないので、新しい考え方を採用した際などの文法の書き直しが困難である。

3　POWER による解決

POWER では記述性の改善を実現するため、手続きを中心にして文法記述を行なうこととした。また手続きを階層的に整理する手法としてかなりの成功を収めているオブジェクト指向の考え方を導入した。その結果として POWER は前述の5つの問題点全てに解決法を与えている。

1 については、継承関係を持つオブジェクトを使用したことにより、例外記述が差分のプログラムで簡単にしかも完全に分離して記述できる。2 に関しては、POWER では各オブジェクトが十分に実行を制御できる。3 についても、解析結果として依存構造風の「関係」による構造を採用したことにより、並列構造や係り受けの交叉も記述できる。4 については、多義の発生箇所を明示的に示す記述形式により、多義を抑制することができ、また多義の発生原因が明確に示される。5 については、POWER では処理のモジュール化が徹底しており、モジュールごと入れ換えることで、新しい手法にも容易に対処できる。

このように POWER では記述性、拡張性の大幅な改善がなされている。

4　POWER の解析機構

4.1　オブジェクト

POWER はオブジェクトと呼ばれる基本単位を用いて解析を進める。オブジェクトは基本的には単語に相当する。そういう意味で単語オブジェクト

POWER: The Word Object based ParsER
Hiroyuki Takahashi, Masahiro Miyazaki
Niigata University

あるいはただ単に単語とも呼ぶ。これらオブジェクトはオブジェクト指向の用語でのオブジェクトと同様のものである。

個々のオブジェクトは固有の変数を持つ。これをクラス変数と呼ぶ。また、メッセージに反応するための固有の手続きを持つ。これをメソッドと呼ぶ。クラス変数の内容はそれを持つオブジェクトのメソッドからのみ変更、参照できる。メソッドの動的な変更はできない。

4.2 メッセージ通信

オブジェクトは他のオブジェクトにメッセージを送り、また他のオブジェクトからメッセージを受け取ることができる。メッセージ通信は往復で一組となっている。メッセージを受けとったオブジェクトは必ず結果を返さなければならない。メッセージには任意の個数の引数を付与できる。結果としては「受理」または「不受理」のいずれかを返さなければならない。また結果に付随して各種の情報を返すこともできる。

メッセージの処理を行なう手続きをそのメッセージに対するメソッドと呼ぶ。メソッドは多相性を有する。つまり、同一のメッセージに対して、オブジェクト毎に異なったメソッドが呼び出される。各オブジェクトは各種のメッセージに対する一群のメソッドを持っており、異なるオブジェクトは異なるメソッドのセットを持つ。

オブジェクト間の相互メッセージ通信において問題となるのは、どのようにしてメッセージ通信の相手を選択するかということである。POWER では二つの方法を提供している。

第一の方法は隣接した単語とのメッセージ通信である。これを隣接メッセージ通信と呼ぶ（図 1)。POWER では各オブジェクトは横一列に配置される。これは文での語順に従って並べられる。メッセージの宛先として「隣接」を指定すると、隣接した単語にメッセージが送られる。この方法では離れた単語とは直接には通信できないので、離れた単語と通信するためには途中の単語にメッセージを中継してもらわなければならない。このことにより単語間の距離による通信可能性の制約が実現される。

第二の方法は送り先オブジェクトの識別子を使う方法である。これを直接メッセージ通信と呼ぶ。これは一般にオブジェクト指向言語で採用されている方法であるが、POWER では二次的な手段である。POWER では全てのオブジェクトに固有の識別子が割り当てられていて、その識別子を宛先に指定することでそのオブジェクトと直接通信できる。

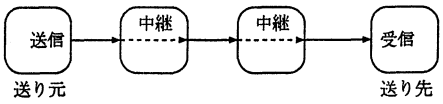

図 1: 隣接メッセージ通信

各オブジェクトは生成時には他のオブジェクトの識別子を知らないので、基本的に隣接メッセージ通信によって通信する。その結果得た識別子によって関連したオブジェクトと直接メッセージ通信を行なうことができる。

4.3 実行制御

POWER はオブジェクトの相互メッセージ通信によって解析を進める。各オブジェクトは勝手に動作するのではなく、一定の実行制御に従って動作する。POWER ではオブジェクトの実行モデルとして、逐次型モデルを採用している。つまり各オブジェクトが処理装置を持ち並列に動作するのではなく、処理装置は一つでそれが割り当てられたオブジェクトのみが動作できる。

処理装置の移動はメッセージ通信にともなって行なわれる。オブジェクト A が別のオブジェクト B に対してメッセージを送った場合、処理装置は B に移動し、B がメッセージを処理して結果を返すと、処理装置は A に戻る。このようにメッセージ通信における処理装置の移動は手続きの呼び出しとほぼ同様のものである。

各単語オブジェクトに最低一回は処理装置が与えられるようにするために、「run」メッセージという特殊なメッセージがシステムから各単語に送られる。システムは左端の（文頭の）単語から順に一つずつに「run」メッセージを送る。この「run」メッセージが解析開始の契機となる。そして全ての単語への「run」メッセージが成功した時点で解析は終了する。「run」メッセージが失敗した単語があったらその解析は失敗する。

4.4 クラス

各オブジェクトはそれ固有の手続きと変数を持ち、メッセージにそれぞれのやり方で反応する。こ

のオブジェクトの記述は一連のメソッドの記述からなる。しかし、あらゆるオブジェクトについてそれぞれメソッドを記述していくとその量は膨大なものとなってしまう。そこで同じような働きをするオブジェクトをまとめて記述することとした。このまとまりをクラスと呼ぶ。例えば名詞の文法的働きはそれが「犬」であれ「机」であれさほど変わらないので、クラス「名詞」としてまとめて記述できる。

各オブジェクトには必ず固有の情報（例えばその表記）がある。そこで、クラス変数という概念を導入し、クラス変数は各オブジェクト毎に固有の値を持つこととした。つまり、同じクラスに属する複数のオブジェクトは手続きは共有するが変数はそれぞれ固有のものを持つわけである。

クラスは継承関係を持つ。継承とは上位のクラスから下位のクラスがメソッドを引き継ぐことである。これによって例えばクラス A に機能を追加してクラス B を作りたいような場合、クラス B には追加機能の部分だけを記述すればよいことになる。例えば「名詞」クラスから「代名詞」クラスを導出したり、「用言」クラスから「動詞」「形容詞」クラスをそれぞれ導出したりできる。また、全てのクラスはクラス「単語」から導出される。

図2にクラスの継承関係による階層の例を示すこの階層は「単語」を根とした木構造になっている。

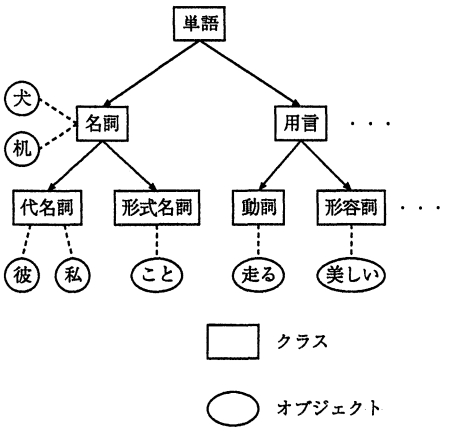

図 2: クラス階層

4.5 解析結果

パーザは必ずその解析結果としてなんらかの構造を作る。この解析結果は翻訳や対話システムなどによって使われる。POWER は関係の集合という形で構造を生成する。任意のメソッドは関係を生成できる。関係は2つのオブジェクトのラベル付けされた順序対である。

一般にパーザの出力には木構造が使われるが、木構造では「うなぎを浜松に食べに行く」のような係り受けの交叉を表現できない。一方、この関係による構造は単語間のあらゆる関係を自然に表現できる（図 3）。

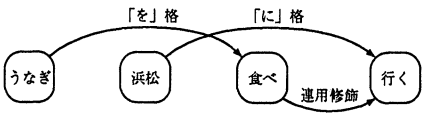

図 3: 解析結果の例

しかし、解析結果の表現形式に制限を与えないことは、一方で処理量の爆発的増加をもたらす恐れがある。しかし POWER では各オブジェクトが実行制御を行なうことができ、あらゆる組合せの全検索はせずに検索木の枝刈りを行なうので処理量の増加を抑えることができる。

4.6 多義

POWER ではルールの選択の問題はないので、それにともなう多義は発生しない。しかし、必要なデータや解析の不足、あるいは本質的な曖昧さのために、メソッドで非決定的な動作をしたい場合がある。POWER では処理分岐命令を用意することで非決定動作を実現している。

処理分岐命令が実行されるとシステムは現在の実行状態をコピーして2つにし、それぞれで異なる処理を実行する。この実行状態をスレッドと呼ぶ。異なるスレッド間で情報がやりとりされることはない。したがって各スレッドは別々に実行できる。また、途中で解析に失敗したスレッドは破棄される。全てのスレッドが失敗したら解析は失敗する。

POWER では多義の発生箇所は処理分岐命令を実行した場所であり、多義の発生箇所が明示的に示される。そのため、多義の発生原因がわかりやす

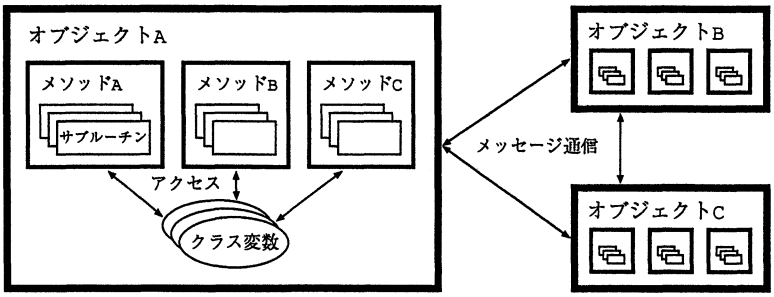

図 4: モジュール構造

く、また多義のしぼり込み手続きをその部分に実装することができる。

4.7 並列処理

ここで、解析の並列処理の可能性について触れておく。オブジェクトモデルは並列処理に向いている。オブジェクトという明確な実行単位があり、各オブジェクトを並列動作させればよいからである。POWER では、並列実行ではなく逐次実行のモデルを採用したが、これは、逐次実行の方が実装が容易であること、並列実行のための安定した標準的プラットフォームがないことなどの現実的な制約による。

並列実行と逐次実行は記述能力の上では大きな違いはないが、並列実行のモデルを用いて並列計算機上で実行させることにより、実行効率を上げることができる。並列実行モデルでは、メッセージの処理中に別のメッセージが来ることがあり得るので、各オブジェクトにメッセージを蓄積するメッセージキューが必要になる。また、メソッドも並列処理を意識した書き方に改める必要があると思われる。しかし、これはオブジェクト間インターフェースの問題であり、オブジェクトの内部的な「判断」の部分には手を付ける必要はないので、比較的簡単に並列処理化が可能である。

4.8 拡張性

POWER は高度の拡張性を実現している。これは、記述のモジュール化が行なわれているからである。

POWER では 3 レベルのモジュール化がなされている。まず第一にオブジェクトレベルのモジュール化がある。各オブジェクトはメッセージの交換以外の通信法を持たないので、メッセージインターフェース（メッセージ名、引数）を変更しない限り、あるオブジェクトを修正する場合に他のオブジェクトのことを考える必要はない。

第二にオブジェクト内部のメソッドによるモジュール化がある。異なるメッセージに対するメソッドは互いにクラス変数を共有している以外は独立している。例えば、名詞が連体修飾を受ける機能と、格要素として動詞と接続する機能はそれぞれ独立したモジュールと考えることができる。

そして第三にメソッド内のサブルーチンによるモジュール化がある。例えば、格パターンのマッチングの部分をサブルーチンとして実装することにより、容易に交換が可能となる。

以上のモジュール構造を図 4に示す。各レベルのモジュールは枠線の太さの違う長方形で表されている。

5 おわりに

POWER はオブジェクト指向の考え方を用いた全く新しいパーザである。従来のパーザは記述性拡張性の問題を抱えていたが POWER はこれらの問題に解決策を与えた。

参考文献

[1] 長尾眞監修:「日本語情報処理」, 電子情報通信学会 (1984)