形式的記述言語パターン解析によるプロトタイプ生成系について

5X－9

橘 賢二　深澤 良彰

早稲田大学理工学部

1　はじめに

形式的仕様記述は、ソフトウエアのライフサイクルにおける仕様記述の段階で誤りを削減するという意味で、有効な手法である。特に、仕様書とユーザの要求との違いをライフサイクルの早期に発見するために、プロトタイプを仕様の段階で作成することも重要である。しかし、プロトタイプ作成のための実現法の詳細を仕様書の中に書くことは、たとえ可能であったとしても、そうすべきではない。それは設計の自由度を阻害してしまうからである。

本研究の目的は、一般的には直接実行が不可能な形式的記述言語において、パターンマッチングによって、実際にどの程度の仕様の直接実行が可能であるかを検証することである。このような研究を行なっている理由は、プロトタイプ実行としては、仕様のすべてを実行する必要はなく、部分的にでも実行できる部分を実行し、実行できない部分に関してはその旨を示すことによって、実用的には十分な効果を得ることができると考えているからである。

本研究では、直接実行不可能な形式的記述言語として、Z言語[1]を採用した。Z言語は数多く存在する形式的記述言語のなかで最も広く使われているものの一つであり、集合論に基づいているので、理解、検証などがスムーズに行なえるなどの特徴があるからである。また、アプリケーションを限定して考えることにした。アプリケーションを限定することによって、その範囲内のアプリケーションの仕様記述法をある程度網羅的に調べることができるからである。我々は、複雑なアルゴリズムを必要としない関係型データベース関連アプリケーションについてだけ着目している。

2　本研究の概略

本研究で作成するシステムは、データベース記述パターンをプログラム生成系に登録し、登録パターンにマッチするZ表記が現われたら、それに見合うプログラムを生成するものである。図1に本プログラム生成系の流れを示す。

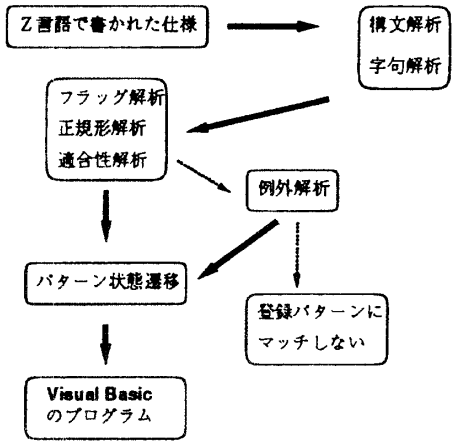

図 1: 本プログラム生成系の流れ

本プログラム生成系の第1段階として、字句解析および構文解析を行なう。これらにはlexおよびyaccを使用している。

第2段階は、命題ごとのにパターンマッチングを行ない、それぞれの命題のパターンを決定する。この手法として以下の3つの手段がある。

1. フラグ解析

   字句解析プログラムから返されるトークンのそれぞれにフラグを立て、その個数だけによって、どのパターンかを限定する。

2. 正規形解析

   正規形解析では、まず、入力された命題に対して数学的に等しいものを自動的につくりだす。この処理により登録パターンを削減することができる。次に、生成された命題に対し、Z言語のオペレータの優先順位に基づいて命題のパターンを決定する。

3. 適合性解析

   適合性解析はマッチするパターンの構成要素の適合性を解析する。

また、これら3つの解析でパターンにマッチしないと判定されたものに対し例外解析を行なう。例外解析には、

Creating the prototype by pattern analysis of formal specification.

Kenji Tachibana and Yoshiaki Fukazawa.

School of Science & Engineering, Waseda University.

フラグ解析などの一連の流れでは解析しにくいパターンが登録してあり、これにマッチしているかを確認する。

第3段階として、パターンとして認識された命題について状態を遷移させていきながら、最終的に生成すべきプログラムを見つける。パターンの状態遷移は、複数の命題で構成されている叙述に対してパターンマッチングを行なう時に効果を発揮する。また、生成されるプログラムには、Microsoft Access のデータベース操作言語である Visual Basic を使用している。

3 実際の例

本プログラム生成系の実際の動きを図2の仕様を例にとって示す。この仕様は Staff という Id, Loc, Salary というフィールドをもつテーブルと、Trainees という Id, Loc, Level というフィールドをもつテーブルがあり、Salary テーブルに Trainees テーブルのうち Loc が「IS」であり、Level が「7」以上であるレコードを追加する SQL における INSERT を示す仕様である [2]。

InsertFunc

$\Delta PersonalDB$

$$Staff' = Staff \cup \{t : Trainees \mid t.Loc = IS \wedge t.Level \leq 7 \bullet StaffRT(t.Id, t.Loc)\}$$

図 2: 入力される仕様記述の例

まず、入力される仕様記述に対し、字句解析、構文解析を行なう。図2の場合、叙述部の命題は以下のように解析される。

$$Staff' = Staff \cup set\_exp$$

set_exp は叙述による集合の定義である。この構文解析された命題に対し、フラグ解析を行なう。フラグは四則演算子、集合演算子、論理演算子などに立てられているので、この場合「∪」および「=」の2つを使用するパターンにマッチする。

次に正規形解析を行なう。正規形解析を行なうためには、まず、数学的に等しい意味をもつ命題を生成する。この場合、以下の4種類の命題が生成される。

$$Staff'\ Staff\ set\_exp \cup =$$
$$Staff'\ set\_exp\ Staff \cup =$$
$$Staff\ set\_exp \cup Staff' =$$
$$set\_exp\ \ Staff \cup Staff' =$$

これらの生成された命題について正規形解析を行なっていく。この場合は最初の命題が登録パターンのうちの一つにマッチする。

次に適合性解析を行なう。この場合はデータベースのテーブルを示す集合 Staff と叙述で定義された集合 set_exp の和集合が、操作後の集合を示す Staff' に等しいので、該当していた登録パターンの適合性が成り立つことが判明する。

最後にパターンの状態遷移を行なう。この場合はその他の命題との関連がないので、状態は変化しない。よって、現在のパターンに対応するプログラムを生成する。他の命題と関連するというのは、

$$A \wedge B$$

のように複数の命題で述語を形成するときのことである。この場合には yacc と lex で生成した別のルーチンを使用して、最終的にマッチするパターンを発見する。

4 評価

現在までのところ、論文 [2] のデータベースアプリケーションに関する仕様の全てとその数学的変換である約 120 種類のパターンに対して、実行可能なプログラムを生成可能である。

次にパターン解析にかかる実行時間をみてみる。SQL における update, insert, delete の3つパターンを解析するのにかか実行時間の１０回の平均をとると、0.875 sec であった。このパターン解析の特徴は、前述のようにフラッグをまず解析することであった。これにより、登録パターンの数が増えても、個々の解析にかかる実行時間にそれほど差ができなくなる。よって、1つのパターンを解析するのに必要な時間は 0.2 - 0.3 sec 必要であると類推された。

5 まとめと今後の課題

従来の Z 言語の実行系は、集合演算を行なって Z 言語を実行しようとするものであった。しかしこの手法では、例えば集合 A から集合 B への関数があったとき、この関数は集合 A のある要素から集合 B のある要素へのマッピングを示しているにすぎず、この記述から実行可能なプログラムを生成することは不可能である。本プログラム生成系では、パターンマッチングとドメイン限定という2つの手法を用いることによって、上記のような実行方法をコンピュータに創造させることの難しい記述に関しても、実行可能なプログラムを生成できる可能性を見い出した。

また今後は、解析できるパターンの数を増やし、パターンマッチングとしての精度を上げるとともに、パターンのマッチング率についても検証していきたい。

参考文献

[1] B.Potter, J.Shinclair and D.Till : An Introduction to Formal Specification and Z, International Series in Computer Science, Prentice-Hall, 1991

[2] D.Edmond : Refining Database System, ZUM'95 : The Z Formal Specification Notation