異なった形式的仕様記述間の変換について

3S-9

FASET(9) ツールゲート

佐藤 茂、及川 健、上枝 泰祐、大林 正晴、川崎 好文、三好 武重、白井 豊
協同システム開発(株)

1. はじめに

協同システム開発(株)では、通産省、IPA 関連事業の一つであるソフトウェア環境統合化技術開発計画(FASET: Formal Approach to Software Environment Technology)を推進している。

現在、FASET では、7種類の仕様記述法に基づく自動化ツールの開発を進めている。筆者らは、それぞれのモデルに合った仕様記述を用いて記述したものを、仕様記述のレベルで一つの記述に変換し、その変換された仕様記述の処理系上で検証をする事を目標に研究を行なっている。

ここでは、FASET での仕様記述法として採用されている関数型言語 K1 から代数的仕様記述言語 ASPELA への変換系と、宣言型仕様記述言語 ML(モジュール機構が付加されている)から ASPELA への変換系を Lisp で作成したので報告する。

2. 問題点及び方針

現在の CASE ツールは、ある一つの記述法に別の記述法を付加する形で色々な記述を可能としている。そのため、記述法が増える度にツールの拡張が必要となっているが、一企業が各記述法に沿ったツールすべてを独立に開発する事は大きな負担になる。一方、協同プロジェクトである FASET では、それぞれの記述ツールを参加企業ごとに開発する必要に迫られていたため、前述のような CASE ツール開発のアプローチは困難であった。そこで、各ツールを独立に開発し、各ツールの橋渡し機構を作る事によって、色々な記述を可能とする事にした。但し、この橋渡し機構を実現するには、以下の課題を解決することが不可欠である。

- 現在7つのツールを独立に作成しているため、各ツール間の橋渡し機構は 7 × 6 = 42 通りとなり、多くの開発費を必要としてしまう。
- 7つのツールはそれぞれモデルが異なっていて、モデルーモデル間の写像を作る事が非常に困難なものがある。

このうち、モデル間変換の部分は、今後の研究課題として捉え、まず類似のモデル間の変換処理を実現することにした。

類似モデルとしては、互に親和性の高い関数型言語のML及び K1 、それと代数型の言語の ASPELAを選んだ。

次の理由により、ML から ASPELA への変換系、及び K1 から ASPELA への変換系を作成した。

- 比較的文法の複雑な方から簡単な方への変換を考える事によって、構文解析として種々のパターンが実験できる。
- 3つの仕様記述言語の中で文法の比較的簡単なものは、ASPELA である。

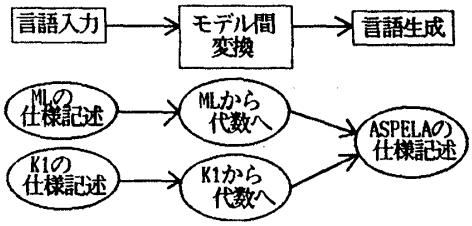

図1 変換系の概念図

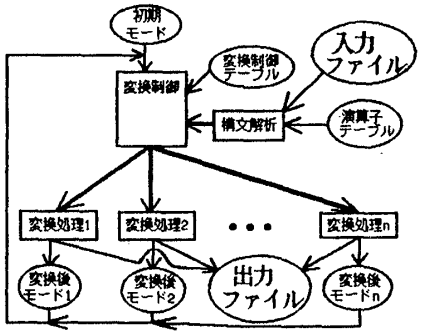

図2変換処理の流れ

変換系を開発するにあたっては、

- モデル間変換に入力される仕様記述は、必ず構文的に正しい事が保証されているので、構文解析が非常に簡略化できる。
- 2つの変換系で共通の構文解析機能、変換処理機能が使えるようにしたい。

等の理由により、テーブル駆動方式を採用して設計を行なった。図2に変換処理と2つのテーブル(演算子テーブル及び変換処理制御テーブル)との関係を示す。また、以下に、その具体的な内容を示す。

(1) 構文解析は、キーワードをすべて演算子と捉え、それらの演算子名、演算子の優先順位、演算子の型からなる以下のような S 式で表現された演算子テーブルを用いて行なう。

```
((<演算子名の文字数>  <演算子名>
   <演算子の型>  <演算子の優先順位>
   [<対をなす演算子名のリスト>])
                ...                )
```

(2) 変換処理は、「文の種類を示すキーワード」、「変換処理関数名」、「変換処理後のモード」からなる以下のような S 式で表現された変換処理制御テーブルを用いて行なう。

```
((<モード名>  <エラー時実行関数>
   <エラー時実行関数実行後のモード>
```

Transformations between different formal specifications
Shigeru Satoh,Takeshi Oikawa,Taisuke Ueeda,Masaharu Ohbayashi,
Yoshifumi Kawasaki,Takeshige Miyoshi,Yutaka Shirai.
Joint System Development Corp.

(<条件名> <実行関数名>
<変換処理後のモード>)
・・・)

3. 変換例

K1 から ASPELA への変換例を以下に示す。

● K1 の仕様記述

```
operator EUCLID ;
   gcd     : integer, integer -> integer .
   gcd2    : integer, integer -> integer .
   bigger  : integer, integer -> integer .
   smaller : integer, integer -> integer .
axiom :
   var x : integer ; y : integer ;.
   gcd( x, y) == gcd2( bigger( x, y),
                       smaller( x, y)) .
   gcd2( x, 0) == x .
   gcd2( x, y) == gcd2( y, ( x mod y)) .
   bigger( x, y) == if x > y then x else y .
   smaller( x, y) == if x > y then y else x .
end EUCLID ;
```

●変換後の ASPELA の仕様記述

```
spec EUCLID ;
sort gcd : number, number => number ;
sort gcd2 : number, number => number ;
sort bigger : number, number => number ;
sort smaller : number, number => number ;
gcd( _x, _y) = gcd2( bigger( _x, _y),
                     smaller( _x, _y)) ;
gcd2( _x, 0) = _x ;
gcd2( _x, _y) = gcd2( _y, mod_1( _x, _y)) ;
bigger( _x, _y) = if_1( greater_than_1( _x, _y),
                        _x, _y) ;
smaller(_x, _y) = if_1( greater_than_1( _x, _y),
                        _y, _x) ;
sort false :  => bool ;
sort true :  => bool ;
sort if_1 : bool, number, number => number ;
sort greater_than_1 : number, number => bool ;
sort mod_1 : number, number => number ;
sort undef_number :  => number ;
if_1( true, _then, _else) = _then ;
if_1( false, _then, _else) = _else ;
greater_than_1( _x#, _y#) = aux( .GT.) ;
mod_1( _x#, _y#) = aux( mod_1) ;
{ end of spec ; }
```

●変換後の補助関数定義（別ファイルに出力される）

```
;;; 補助関数定義
(defaux .GT.
       (LAMBDA (X Y)
         (COND
           ((NUMBERP X)
               (IF (> X Y) 'TRUE 'FALSE))
           ((CHARACTERP X)
               (IF (CHAR> X Y) 'TRUE 'FALSE))
           (T (IF (STRING> X Y) 'TRUE 'FALSE)))
       ))
(defaux mod_1
       (LAMBDA (X Y)
         (IF (AND (NOT (ZEROP Y))
                  (INTEGERP X) (INTEGERP Y))
             (MOD X Y) 'UNDEF-NUMBER)))
```

4. 検討

(1) 構文解析に関して：

演算子テーブルを用いた構文解析は、言語間変換に一般的に使えそうであるが、以下の点に注意を要する。

・文の終端の検出（継続行等の取り扱い）
文をどこで切るかは、言語ごとに対応せざるをえない。

・入れ子の処置
if then else のように、入れ子になるようなキーワードに対しては、演算子テーブルに<対をなす演算子名のリスト>を付加する事によって入れ子の構文解析を可能としている。その為以下に示す if then else の演算子テーブルの(if then (else))のようなリストを演算子テーブルに付加しなければならない。

```
((2 "if" fxyy 3000 (if then (else)))
 (4 "then" gxfyy 3200 (if then (else)))
 (4 "else" gyxfy 3100 (if then else)))
```

・文の種類によるキーワードの意味の違い
ML では、文の種類によりキーワードの優先順位を変える必要がある。その場合、「キーワード」と「それに対応する演算子テーブル」を要素とするテーブルを用意し、ダイナミックに演算子テーブルを変更できるようにしなければならない。

・この構文解析方法では非常に難しい構文
この場合、構文解析の難しい構文のキーワードが出て来た処で構文解析を途中で打ち切り、打ち切ったキーワードの変換処理の中で対応出来るようにしなければならない。

(2) 変換処理に関して：

変換処理制御テーブルと演算子テーブルの定義等、キーワードに対応するテーブルを変更するだけで変換処理ができるとの想定で設計を進めたが、以下の点で若干の修正を必要とした。

・演算子テーブルを用いた構文解析では困難な構文の場合、変換処理の中で、構文解析を呼び、その結果を取りまとめて、変換処理を行なわなければならない。

例： ML に於ける let [dec] in [exp] end

・この方法では、一方の言語にあって、もう一方の言語にない機構は、意味を考えない変換では困難である。
例えば、ML の例外処理や高階関数は、ASPELA には無い仕様なので変換できなかった。

5. 今後の課題

4. 検討 (2) で述べられている様に類似のモデル間に於ても変換が困難な部分があるが、その部分が変換可能化かどうか、もっとモデル間変換を研究する必要性がある。また、次の様なモデル間の変換も扱えるように、より一般的な変換方法を確立する研究も必要である。

状態遷移、 代数、 データフロー、
ER モデル、 自然言語、 論理、
フレーム型（オブジェクト指向を含む）、
ペトリ・ネット

参考文献

(1) ソフトウェア環境統合化技術開発計画テクニカル・レポートNo.1、協同システム開発株式会社
(2) ソフトウェア環境統合化技術開発計画テクニカル・レポートNo.2、協同システム開発株式会社