情報処理学会第37回(昭和63年後期)全国大会

仕様記述言語向きの変換系記述言語ＴＤＬ

2L-7

尹　仙淑　　栗野俊一　　廣瀬　健　　深澤良彰
(早稲田大学理工学部)

1．はじめに

形式的仕様記述言語で書かれた仕様を、実行可能なプログラムに変換し、これをプロトタイプとして利用することが要求されている。このための変換系記述言語ＴＤＬ（Translator Description Language）を定義し、この言語記述から変換系を自動生成するシステムを作成した（図１参照）。

仕様記述言語は、厳密性とともに、読みやすさが重要であり、読みやすさは記述対象とする分野に大きく依存する。特定の分野の適用業務に対して、可読性・記述性の高い仕様を実現するには、仕様記述言語の表現形式をその分野に対して適合させる必要がある。

そのためには、仕様記述から実行可能形式へ変換するための処理系を容易に変更できるようにしなければならない。ＴＤＬは、我々が開発してきた仕様記述言語ＷＳＮ（Waseda Specification Notation）[1]やOxford大学とＩＢＭで開発された仕様記述言語Ｚなどに、適用業務向きの機能を付加した言語による記述を、prologなどの実行可能なプログラムに変換する変換系を記述する言語である。

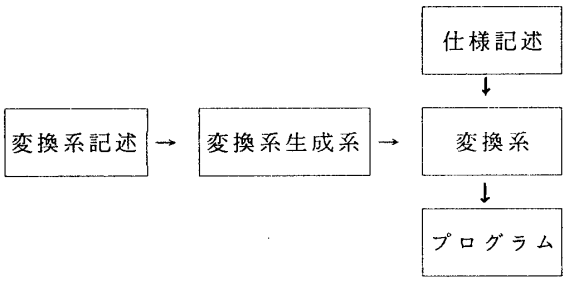

図１　本システムの基本構造

2．変換系記述言語ＴＤＬ

2．1　ＴＤＬの特徴

ＴＤＬの設計にあたっては、以下の５項目を重視している。

１）変換系記述を読みやすくするために、構文と意味をともに関数的に記述にした。

２）高い記述性を実現するために、入力言語の文法と出力言語の文法と入力言語の意味を同時に書けるようにした。

３）記述の独立性を増すために、入力言語の構文要素と対応する出力言語の構文要素を構文規則ごとに記述した。

また、記述性を増すために、

４）入力言語の１つの構文規則に現れる構文要素の順序に対し、対応する出力言語の構文要素の順序を交換できるようにした。

５）出力言語の構文が文脈自由言語であっても、区切り記号、句読点等の記述は、文脈依存的となることがある。そのために、区切り記号、句読点等の出力を制御する演算子を追加した。

2．2　ＴＤＬの記述法

入力言語（仕様記述言語）から出力言語への変換は、入力言語の文法によって定義される構文要素から出力言語の文法によって定義される構文要素への変換と見なせる。よって、ＴＤＬにおいては、変換系の記述を、入力言語の構文要素と出力言語の構文要素間の対応関係の関数的な記述によって表現する。そのために、それらの間の変換関数を定義する。

１）ＴＤＬによる記述の考え方

まず、入力言語の構文要素に対応させる出力言語の構文要素を決める。ここでは、〈in〉に対して〈out〉を対応させることとし、この関係を関数ｔで表すこととする。

〈out〉 ＝ ｔ(〈in〉)

その後で、それらの構文要素を定義する構文要素間の対応を定める。いま、〈in〉と〈out〉が各々次のように定義されていたと仮定する。

〈in〉 ::= 〈in 1〉 〈in 2〉 〈in 3〉
〈out〉 ::= 〈out 1〉 〈out 2〉 〈out 3〉

また、〈in i〉,〈out i〉（i = 1,…,3）の関係が変換関数ｔ1～ｔ3によって、以下のように、対応を付けられたとする。

ｔ2(〈in 2〉) = 〈out 1〉
ｔ1(〈in 1〉) = 〈out 2〉
ｔ3(〈in 3〉) = 〈out 3〉

このとき、変換関数ｔをＴＤＬでは以下のように定義し、記述する。

ｔ (〈in〉) = ｔ(〈in 1〉 〈in 2〉 〈in 3〉)
= ｔ2(〈in 2〉) & ｔ1(〈in 1〉)
& ｔ3(〈in 3〉)

このような記述は非終端記号ごとに得られ、これを記述単位と呼ぶ。

Translator Description Language TDL for Specification Languages
Sunsook Yoon, Syunichi Kurino, Ken Hirose, Yoshiaki Fukazawa
Waseda University

2）ＴＤＬによる記述例

図２にＴＤＬによる記述例を示す。この例は、ＷＳＮの関数定義（〈function macro〉）という構文要素からPrologのルール（〈rule〉）という構文要素への変換規則を記述したものである。

図２の(ａ)と(ｂ)において、入力言語の構文を定義する。この例における関数定義を通常のＢＮＦで表現すると、下記のようになる。

〈function macro〉 ::= 〈function name〉
'(' 〈term seq〉 ')' '=' 〈term〉 …(1)

このように、入力言語の構文定義は、ほぼ通常のＢＮＦ通りに記述すればよい。

図２の(ａ)と(ｃ)において、入力言語の意味を出力言語の構文要素を組み合わせて記述する。しかし、出力言語の構文要素のみでは表現できないもの、表現しにくいものがあり、これらに対処するためにいくつかの組込み関数が用意されている。図２の(ｃ)では、predname、gen、take_var、take_predがこの組込み関数であり、それぞれ次のような意味をもつ。関数prednameは、入力言語（この場合にはＷＳＮ）における関数名を、出力言語（Prolog）の述語名に適合するように変換する。関数genは、与えられた文字列を出力する。一般には、入力言語の構文要素と出力言語の構文要素は１対１に対応する。しかし、たとえば、ＷＳＮの項（〈term〉,〈term seq〉）の場合には、prologの複数の構文要素（項と述語）に対応させる必要はある。それは、項の評価を、prologでは述語を使って行なうためである。従って、項と述語を中間形式として１つにまとめた対応関係にしておき、出力の際には別個に扱う。そのために、組込み関数take_pred、take_varが用意されている。take_predは述語を、take_varは項を取り出す。

termとterm_seqはＷＳＮの構文要素〈term〉と〈term seq〉についての変換関数で、別に定義される。

```
func_macro( 〈function macro〉 )                    (a)
= func_macro( 〈function name〉                     ┐
      '(' 〈term seq〉 ')' '=' 〈term〉 )            ┘(b)
= gen( predname( 〈function name〉 ) )          &   ┐
  gen( '(' )                                   &   │
  gen( ( take_var( term( 〈term〉 ) )           &   │
  gen( ',' ) / gen( take_var( term_seq               │
                  ( 〈term seq〉 ) ) )          &   │(c)
  gen( ')' )                                   &   │
    gen( ':-' ) /                                  │
     ( gen( take_pred(term(〈term〉)) )         &   │
      gen( '&' ) /                                 │
        gen( take_pred( term_seq                   │
                    (〈term seq〉) ) )              │
    )                                          &   ┘
  gen( '.' )
```

図２　〈function macro〉の変換記述

入力言語の構文と出力言語の構文の対応関係が複数対１であるときは、出力言語を変えないように、その構文規則を変形する必要が生じる。

この例では、前述のように(1)の構文要素〈term〉と〈term seq〉が各々２つ構文要素に対応する。そこで、通常のprologのルール（〈rule〉）である(2)とは１対１に対応関係させることができない。よって、(2)を(3)のよう変形する。

〈rule〉 ::= 〈head name〉
'(' 〈argument seq〉 ')'
':-' 〈body seq〉 '.' …(2)

〈rule〉 ::= 〈head name〉
'(' 〈var〉 ',' 〈arguments〉 ')'
':-' 〈body〉 '&' 〈bodies〉 '.' …(3)

(1)の構文要素〈term〉は(3)の構文要素〈var〉と〈body〉に対応し、その関係は次のようになる。

gen((take_var(term(〈term〉))) = 〈var〉
gen(take_pred(term(〈term〉))) = 〈body〉

(1)のすべての構文要素に対して対応関係を記述すると図２の(ｃ)が得られる。

拡張ＢＮＦで使用できる演算子はすべて構文定義、意味定義において使用できる。これに加えてＴＤＬでは、出力言語の区切り、句読点等の出力を制御するのに使う演算子／が導入されている。この演算子の記述法とその意味を以下に示す。

$$a / \alpha = \begin{cases} a\,\alpha & \alpha \neq \varepsilon \\ \varepsilon & \alpha = \varepsilon \text{（ただし}\varepsilon\text{は空列）} \end{cases}$$

実際にＴＤＬを用いて、各種の変換系を記述してみると、この演算子の使用頻度は非常に高い。

３．おわりに

仕様記述言語向きの変換系記述用言語ＴＤＬの仕様と記述法について述べた。

今後ＴＤＬの記述能力を確認するために、種々の仕様記述言語の変換系をＴＤＬにより記述することを考えている。また、ＴＤＬの記述性を高めるために、その機能を拡張することも考えている。ＴＤＬ自身も仕様記述言語であり、その開発にＴＤＬを利用することによって、機能の拡張は容易にできる。

謝辞

本発表は、早稲田大学情報科学研究教育センター「ソフトウェア開発における仕様記述あるいは検証の実用性に関する研究」部会の研究成果の１つであり、この研究活動に参加あるいは支援してくたさった方々に心より感謝致します。

参考文献

（１）河野他：ＷＳＮによるスケジューラの詳細仕様記述とその経験，Bulletin of Centre for Informatics, Waseda Univ., Vol.5, 1987.