2L-8

仕様記述言語向きの変換系記述言語 ＴＤＬの適用と効果

栗野俊一　尹　仙淑　深澤良彰　廣瀬　健
(早稲田大学理工学部)

１．はじめに

プロトタイピングを目的とした、形式的仕様記述から実行可能なプログラムへの変換系の作成が要求されてきている。この作業を支援するために、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）。

本稿では、このシステムを実際に幾つかの仕様記述言語に適用した際の使用経験について述べる。

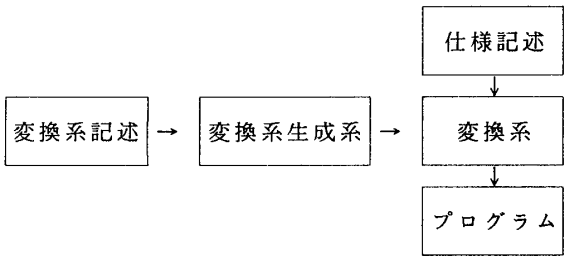

図１　本システムの基本構造

２．ＴＤＬ

ＴＤＬは、ＤＣＧ(Definite Clause Grammar)の一種の変形である。入力言語の構文規則を記述し、その構文規則を満たすテキストの入力に対し、どのような処理を行なうかを記述する。ここで、その処理を出力言語のテキストの出力とするならば、それは変換記述になる。ＴＤＬでは、これを入力テキストから出力テキストへの関数として記述する。ＴＤＬは、この仮定の下に、入力言語の構文のクラスをＬＬ（ｋ）に制限する一方、より変換記述に適した拡張が行なわれている。

ＴＤＬによる記述は、変換する入力言語の構文規則と、出力言語の構文規則の一対一の対応を表現している。このため、必要に応じて、対応を一対一にするように、出力言語の文法を変形する。この様にして、その記述を、各々の生成規則単位でおこなうので、モジュラリティが高い。その一方、識別子の管理表などの大域的な情報に関しては、組込関数に依存しなければならず、記述性に乏しいという欠点をもつ。

３．仕様記述言語

我々は、ＴＤＬを次の様な仕様記述言語に適用した。ただし、出力言語は全て prolog である。

（１）ＷＳＮ

ＷＳＮ[1]は set theory base, week second order logic 等のパラダイムから設計されており、述語論理を基本にしている。なお、実働化の際には、実行のために必要な意味上の制限をＷＳＮに対し設けている。

（２）ＴＤＬ

ＴＤＬもまた、変換系の仕様記述言語と解釈するならば、それ自身ＴＤＬにおける記述の対象と成り得る。実際に我々は、ＴＤＬから prolog への変換系をＴＤＬで記述した。

（３）Ｚ

Ｚ[2]は、schema記述をその中心としている仕様記述言語である。Ｚの実働化は、変換系と実行系の組合せで行なった。ここでも、ＷＳＮと同様、実行のための制限を設けている。

４．適用結果

これらの結果、次の様な評価が得られた。

（１）生産性

ＴＤＬとＷＳＮの二つの変換系の開発は、ほぼ並行に行なった。つまり、ＷＳＮの変換系の記述に不足な点、不便な点が生じた時点で、ＴＤＬの言語仕様とその変換系が拡張された。ＴＤＬの変換系は、ブートストラップ技法を用い、ＴＤＬで記述したため、それらの拡張は容易であった。

（２）記述性

変換系の記述を、『始めに入力言語の意味単位と対応する構文要素だけに着目し、その変換スキーマを記述する。その後に、他の部分を肉付けして行く。』とする方針をとることにより、変換系の記述が容易にできることがわかった。

（３）可読性

拡張ＢＮＦの記法を採り入れ、関数的に記述できるため可読性に優れている。しかし、後で述べるように、大域的な情報の操作など、ＴＤＬの枠組みで扱えない部分の記述が必要になり、それが複雑に混入した場合には、可読性が低下した。

Experiences and Evaluation of Translator Description Language TDL
Syunichi Kurino, Sunsook Yoon, Ken Hirose, Yoshiaki Fukazawa
Waseda University

（4）文脈依存記述

ＴＤＬでは、識別子の宣言など、文脈依存に関する記述を、大域的な情報操作の形で行なう必要がある。

たとえば、ＷＳＮの変換系では、変数の束縛がネストし、束縛変数に関して、ブロック構造を形成するが、 prolog には対応する構文が存在しない。したがって、変数名の一意性を保証するために束縛変数の表が必要になる。また、ＴＤＬの変換系では、非終端記号名を管理する必要が生じた。

ＴＤＬの設計思想から、この問題は、避けることができない。しかし、大域的な情報操作のための技術は確立しており、この部分は、ライブラリー化可能である。しかも、この大域的な情報操作に関する記述の他は、ほぼ、変換スキーマになるため、この記述は後に独立して追加すれば良い。

（5）適用範囲

ＴＤＬにおける、純粋な構文構造の変換という枠組みだけでは、対応できない部分がある。

たとえば、ＷＳＮの項は prolog で、項の評価を行なう述語に変換される。このために、ＷＳＮの一つの構文要素に対し、 prolog の相異なる複数の構文要素（項と述語）が対応する。その結果、入、出力言語の構文要素（規則）が一対一に対応しなくなる。また、ＴＤＬの変換系においては、繰返し記述を prolog の再帰に変換するため、繰返し記述に対し、新たな述語と、その述語を頭部とするルールの二つを対応させる必要が生じた。この、項と述語、述語とルールは、定義記述と、その定義の参照の関係であり、連続しない位置に記述される。この定義と定義参照の記述位置の関係は、一般的なものであり、他の言語を出力に選ぶ場合にも生じる。しかし、逆に、定義部分とその参照位置が連続するような言語を出力言語とするか、または、それを中間言語とし、さらに、定義部分を適切な位置に集めるようにすれば回避できる。実際に我々は、一種の中間形式を許すことによって、これに対応した。まず、中間形式を表現する記法を導入し、複数の、異なる構文要素のテキストをまとめて一つのものとして扱えるようにし、形式的に対応を一対一にした。次に、出力言語へのテキストの出力を一時的に切替ができるようにし、それぞれ別の適切な位置に置けるようにした。

（6）出力整形記述

出力言語の構文が文脈自由言語であっても、区切り記号や、終止記号等の記述には、多少、文脈依存的な記述を許す必要がある。ＴＤＬでは、出力言語の構文記述に単純な演算子を導入し、構文記述の枠組みでの記述を可能にしている。

（7）効率

プロトタイピングを行なうための変換系の作成が目的であることから、変換系の作成するプログラムの質は問題にならなかった。むしろ、変換系の実行速度の方が、プロトタイピングのサイクルに含まれるため、問題になった。その原因は、変換系生成系の記述にあるが、これは、ＴＤＬで記述されているため、これを最適化することにより対処可能である。

5．副次効果

ＴＤＬをいくつかの仕様記述言語に適用した結果、次の様な副次効果も得られた。

①ＴＤＬの変換系においては、プリプロセッサの部分を独立させて記述した。この様に変換系を幾つかの変換系の組合せとして記述することにより、変換系に柔軟性を持たせることが可能である。

②Ｚの実働化は、変換系と実行系の組合せで行なわれた。この様に、一方で中間言語の実行系を用意し、ＴＤＬによって、中間言語への変換系を作成することにより、短期間で実働化可能になりうる。そして、実行系での評価のうち、静的に評価可能であるものを変換記述に移行させることにより、段階的に変換系を完成させることができる。

③ＴＤＬ記述の内の変換スキーマの部分を、入、出力言語間の等価変換としてとらえることにより、変換系の検証の可能性が生じる。

6．おわりに

変換系記述用言語ＴＤＬを実際の形式的仕様記述言語に適用した経験について述べた。ＴＤＬは、変換系記述用言語として、単純な基本原理に基づいているため、記述が容易になり、プロトタイピングをその目的とする変換系の記述、作成に適していることが確かめられた。また、その単純性のため、仕様記述と生成された実行可能プログラムとの等価性の検証も容易になると考えられる。

今後、未解決点の対応と、これらの変換系の等価性の検証を行なう予定である。

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だ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.

［２］J.M.Spivey: Understanding Z, Cambridge University Press, 1988.