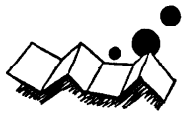
解　説



構文と意味の記述によるプログラムの作成†

森　沢　好　臣††

1. はじめに

共通例題・酒倉庫問題に実行可能な仕様作成という立場から解答を与える.

プログラム言語の定義はふつう構文記述と意味記述から構成されている. 構文記述については BNF (Backus-Naur Form) がよく普及している. 意味記述については従来は自然言語に依存していたが, 最近は処理系の自動生成を目指して属性文法や表示的意味記述が用いられはじめている[1),2)].

事務処理問題については, データ構文記述からプログラム構造を導出する Jackson 法[3)]が普及している. Jackson 法では構文記述をもとにプログラムを導出していったが, この手法を一歩進めて, 事務処理問題もプログラム言語の定義方法にならって構文記述そのものと意味記述がプログラムの仕様であり, かつそのままで直接実行させるという風にすることができる.

ここではつぎのような手段を採用する. まず課題の入出力データの構文部分を BNF によって記述し, これに入出力データ間に成立すべき関係などの意味部分を付加したものを DCG (Definite Clause Grammar)[4),5)]によって記述する. これを仕様と考える. DCG は機械的に Prolog に変換できる (DCG トランスレータを持つ Prolog 処理系もある) ので, 変換された仕様は実行可能なものになる[6),7)]. すなわち, 変換された Prolog プログラムが, Prolog の計算機構によって下降型構文解析／文生成プログラムとして動作するので, 入力データの解析や出力データの合成が行えるのである.

2. 課　題

スペースの都合上, 次のように簡略化した共通問題を考える. 図-1 を参照のこと.

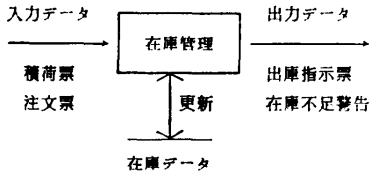


図-1　酒倉庫在庫管理

入力データは積荷票（コンテナ番号, 酒名, 数量）と注文票（注文番号, 注文主, 酒名, 注文数量）である. 出力データは注文票に対する出庫指示票（注文番号, 注文主, (コンテナ番号, 酒名, 数量, コンテナ空マーク）の繰返し）または在庫不足時の在庫不足警告（注文番号, 注文主, 酒名, 注文数量）である. 積荷票が入力されるとコンテナと酒単位の在庫データを更新する. 一件一酒銘柄の注文票が入力されると在庫データを調べ, 在庫がないか在庫量が注文数量を満たさないときは在庫不足警告を出力する. 在庫量が注文数量を満たすと, 出庫指示票を出力する. 出庫指示票ではコンテナごとに出庫指示がなされ, 空になる予定のコンテナであればコンテナ空マークを合わせて出力する. 入力データの妥当性検査やコンテナを早く送りだすための工夫などは行わない.

3. プログラムの仕様記述

3.1 入出力データの構文記述

まず入出力データの具体構文を BNF によって記述する. 図-2 および 図-3 がそれである.

出力データについての構文記述は, 出力結果を見やすくするために空白の挿入とか行換えの指示等の非本質部分が混在して多少読みにくくなっている. 入力データの注文票の繰返しに対応して出力構文〈output〉が繰り返され, 注文票が一件入力されればそれに対応して一件の〈output〉の処理ができるから, 図-3 の出力データ構文において [..] で囲まれた構文は出力側

† A Step toward a Walkable Specification by Yoshitomi MORISAWA (Software Planning & Control, Product Planning & Control, Nippon UNIVAC Kaisha, Ltd.).

†† 日本ユニバック(株)プロダクト本部ソフトウェア企画部

```
<input>     ::= <trans>
<trans>     ::= <tran> <trans> |
                <empty>
<tran>      ::= <container> |
                <order>
<empty>     ::= ''
<container> ::= 'CONT' <c_no> <sake> <qty>
<c_no>      ::= <integer>
<sake>      ::= <identifier>
<qty>       ::= <integer>
<order>     ::= 'ORDER' <o_no> <orderer> <sake> <qty>
<o_no>      ::= <integer>
<orderer>   ::= <identifier>
```

図-2 入力データ構文

```
[ <output_data> ::= <outputs>                                    ]
[ <outputs>     ::= <output> <outputs> |                         ]
[                   <empty>                                      ]
  <output>      ::= <shipping_instruction> |
                    <out_of_stock_notice>
  <shipping_instruction> ::= 'SHIP ' <s_o_no> <s_orderer> <n_l>
                             <s_items>
  <s_o_no>      ::= <integer>
  <s_orderer>   ::= ' ' <identifier>
  <s_items>     ::= <s_item> <s_items> |
                    <empty>
  <s_item>      ::= <s_c_no> <s_sake> <s_qty> <empty_mark> <n_l>
  <s_c_no>      ::= ' Container ' <integer>
  <s_sake>      ::= ' ' <identifier>
  <s_qty>       ::= ' ' <integer>
  <empty_mark>  ::= '(Empty)' |
                    ' '
  <out_of_stock_notice> ::= 'OUT_OF_STOCK '
                            <unalloc_o_no>
                            <unalloc_orderer>
                            <unalloc_sake>
                            <unalloc_qty> <n_l>
  <unalloc_o_no>    ::= <integer>
  <unalloc_orderer> ::= ' ' <identifier>
  <unalloc_sake>    ::= ' ' <identifier>
  <unalloc_qty>     ::= ' ' <integer>
  <n_l>             ::= 'CR/LF'
```

図-3 出力データ構文

DCGに変換する時省略する．したがって出力側DCGは出庫指示票及び在庫不足警告の構文のみが変換される．このように出力データ構文を縮小することによって意味記述のために導入する属性の数を減少させ，より自然な解にすることができる．

3.2 意味の記述

3.2.1 補助関数（述語）の準備

属性文法のやり方にならって，在庫状況を示す状態空間を考え，これを媒介して意味記述を行うことにする．この状態空間Stとしてはコンテナ番号全体Contと酒名全体Sakeとの直積空間 Cont×Sake を定義域とし，酒数量全体Qtyを値域とする関数 Cont×Sake →Qty（$(c,s)\to q$：コンテナcに収納されている酒名sの数量q）の集合を考えればよい．この関数が入力データによってどう更新されるかを考えることになる．この状態空間 St を Prolog を用いてつぎのように実現する．

St を3つ組 stock (c,s,q) の集合とする．Stの初期状態は空集合ϕとする．積荷票の入力に際して，それから定まる c,s,q により3つ組 stock (c,s,q) を定め，St $\cup\{\text{stock}(c,s,q)\}\to$St を行う．注文票の入力に際しては，それから誘導される order (s,q) によって St を更新する．またある Co∈Cont に対して St(Co)$=\{\text{stock}(c,s,q)\in\text{St}\mid C=\text{Co}\}$ で，St(Co)のqたちの総和が0になれば St-St(Co)→St を行う．

以上の St への追加，変更，削除をそれぞれ add_stock (C,S,Q)，change_stock (C,S,Q,NQ) および delete_stock (C) の述語によって行うようにする．

付随する補助述語としてつぎを用意する．

sake_stock (S,T)：倉庫内にある酒名Sの総量T

is_empty_cont (C)：コンテナCは空

以上の Prolog プログラムは図-4 のとおりである．

ここで，

bagof(X,P,S)：SはPを満足するすべてのXの

```
add_stock( C,S,Q )       :- assert( stock( C,S,Q ) ).
change_stock( C,S,Q,NQ ) :- retract( stock( C,S,Q ) ),
                            assert( stock( C,S,NQ ) ).
delete_stock( C ) :- retract( stock( C,_,_ ) ), fail.
delete_stock( _ ).
sake_stock( S, T ) :- bagof( Q, C^stock( C,S,Q ), L ),
                      total( L, T ).
sake_stock( S, T ) :- T is 0.
is_empty_cont( C ) :- bagof( Q, S^stock( C,S,Q ), L ),
                      total( L, T ), !, T = 0.
total( [], T )     :- T is 0.
total( [X|Y], T ) :- total( Y, New_T ), T is X + New_T.
total( X, T )      :- T is X.
```

図-4　補助述語

```
input --> trans.
trans --> tran, !, trans.
trans --> empty.
tran  --> container |
          order.
empty --> "".
container --> ['CONT'], c_no( C ), sake( S ), qty( Q ),
              { add_stock( C, S, Q ) }.
c_no( C ) --> [C].
sake( S ) --> [S].
qty( Q )  --> [Q].
order --> ['ORDER'], o_no( N ), orderer( O ), sake( S ), qty( Q ),
          { make_output( N, O, S, Q ) }.
o_no( N ) --> [N].
orderer( O ) --> [O].
```

図-5　入力側 DCG

つくる多重集合*である.

Y^Q：　Qを満足するYが存在する.

assert (C)：　集合（データベース）にCを追加する.

retract (C)：　集合からCを削除する.

という Prolog 組込みの述語を使用する.

3.2.2 意味の記述

BNF を用いた構文記述をもとに，これに入出力データ間に成立すべき関係などを付けて意味記述を完成させる．このとき DCG を用いて書けば図-5 および図-6 のようなプログラム部分が得られる．意味を記述するために構文記述に付けられた属性を説明する.

C：コンテナ番号

S：酒名

Q：数量または注文数量

N：注文番号

O：注文主

T：酒名Sの総量

U：出庫指示票作成のための注文数量残

NU：同上（更新用）

OQ：注文票の注文数量

SQ：出庫指示票内の数量

NQ：在庫データ更新のための数量

以上を，Prolog プログラムとして完成させるために，若干段取り（house-keeping）部分(図-7)を補足追加する．段取り部分では，

see (F)：　入力ファイルFを開く.

seen：　入力ファイルFを閉じる.

get (K)：　現在の入力ファイルより空白でない一文字をKに読み込む.

get0 (K)：　現在の入力ファイルより一文字をKに読み込む.

\+P：　not P

という Prolog 組込み述語を使用する.

以上の仕様は Prolog プログラムに変換することによってプログラムとみなすことができる．この仕様がどのように動くかを図-5 及び図-8 を参照しながら簡単な例を用いて説明する.

stock_control(Fname) が呼ばれると，まず get_file Prolog プログラムが呼ばれる．get_file は，Fname で指示された入力ファイルより入力データを読み込み，入力データをトークン・リストに変換する．変換したトークン・リストを，get_file の属性 In に入れる．今，Fname によって指示された入力ファイルよ

* バッグ (bag)，疎な和あるいは分離和 (disjoint union) などといわれる.

```
output( N,O,S,Q ) --> { sake_stock( S, T ) },
       ( ( { T >= Q }, shipping_instruction( N,O,S,Q ) ) |
         ( { T <  Q }, out_of_stock_notice( N,O,S,Q ) )   ).
shipping_instruction( N,O,S,Q ) -->
       { write( 'SHIP ' ) }, s_o_no( N ), s_orderer( O ), n_l,
       s_items( S,Q,U ).
s_o_no( N )      --> { write( N ) }.
s_orderer( O )   --> { write( ' ' ), write( O ) }.
s_items( S,Q,U ) --> { Q \== 0 }, s_item( S,Q,U ), !,
                                  s_items( S,U,NU ).
s_items( _,_,_ ) --> [].
s_item( S,OQ,U ) -->
     { stock( C,S,Q ), Q > 0,
       ( ( Q  > OQ, SQ is OQ, U is 0,      NQ is Q - OQ ) |
         ( Q =< OQ, SQ is Q,  U is OQ - Q, NQ is 0      )  ),
       change_stock( C,S,Q,NQ )  },
     s_c_no( C ), s_sake( S ), s_qty( SQ ), empty_mark( C ), n_l.
s_c_no( C ) --> { write( ' Container ' ), write( C ) }.
s_sake( S ) --> { write( ' ' ), write( S ) }.
s_qty( SQ ) --> { write( ' ' ), write( SQ ) }.
empty_mark( C ) --> { is_empty_cont( C ), delete_stock( C ),
                      write( '(Empty)' ) }.
empty_mark( _ ) --> { write( ' ' ) }.
out_of_stock_notice( N,O,S,Q ) --> { write( 'OUT_OF_STOCK ' ) },
                                   unalloc_o_no( N ),
                                   unalloc_orderer( O ),
                                   unalloc_sake( S ),
                                   unalloc_qty( Q ), n_l.
unalloc_o_no( N )    --> { write( N ) }.
unalloc_orderer( O ) --> { write( ' ' ), write( O ) }.
unalloc_sake( S )    --> { write( ' ' ), write( S ) }.
unalloc_qty( Q )     --> { write( ' ' ), write( Q ) }.
n_l --> { nl }.
```

図-6 出力側 DCG

```
stock_control( Fname ) :- get_file( Fname, In ),
                          input( In, [] ).
make_output( N, O, S, Q ) :- output( N, O, S, Q, Temp, [] ).

get_file( F, L ) :- see( F ), tokens( L ), seen.
tokens( [T|N] ) :- token( T ), !, tokens( N ).
tokens( [] ).
token( T ) :- get( K ),
              ( ( isdigit( K ),  digits( U )    ) |
                ( isletter( K ), alphanums( U ) )   ),
              name( T, [K|U] ).
isdigit( X )  :- \+( atom( X ) ), X > 47, X < 58.
isletter( X ) :- \+( atom( X ) ),( ( X > 96, X < 123 ) |
                                  ( X > 64, X < 91  )  ).
digits( [K|U] ) :- get0( K ), isdigit( K ), !, digits( U ).
digits( [] ).
alphanums( [K|U] ) :- get0( K ), alphanum( K ), !,
                      alphanums( U ).
alphanums( [] ).
alphanum( K ) :- isletter( K ) |
                 isdigit( K ).
```

図-7 ハウスキーピング・プログラム

り "CONT 200 Asahi 600 ORDER 150 NUK Asahi 700" というデータが入力されたと仮定する．get_file によって〔CONT, 200, Asahi, 600, ORDER, 150, NUK, Asahi, 700〕というトークン・リストに変換され，get_file は終了し，次に入力側 DCG，input が呼ばれる．この時解析すべきトークン・リスト及び解析の終了条件のリストは，DCG 表現では陽に記述されない．入力側 DCG の動きを簡単に説明すると，まず input が呼ばれると input より trans が呼ばれる．trans より tran が呼ばれる．tran より container が呼ばれる．container で入力データのトークン・リストが "CONT" で始まるか否かを調べる．この場合，"CONT" で始まるので，c_no を呼び出して C に 200，sake を呼び出して S に Asahi，qty を呼び出して Q に 600 を得る．そして，在庫データ・ベースに登録するために add_stock (200, Asahi, 600) を呼び出す．この登録によって container の処理が終了し，よって tran の処理が終了し，次の trans が呼び出される．

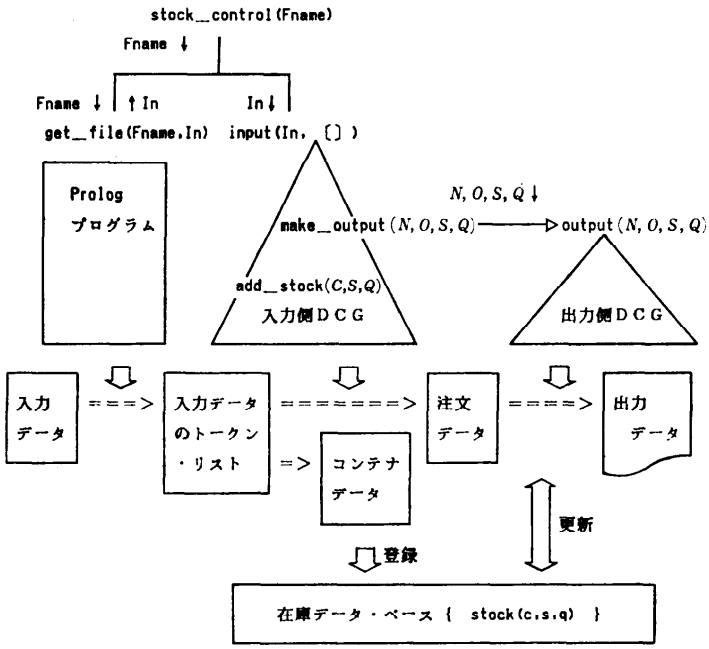


図-8　酒倉庫在庫管理プログラム

次の trans より tran, そして container と呼ばれる. container で入力データのトークン・リストが "CONT" で始まるか否かを調べる. この場合, "ORDER" で始まるので container は失敗し, 次の候補である order が呼ばれる. o_no, orderer, sake そして qty を順次呼び出して N, O, S そして Q の値を得て, make_output (150, NUK, Asahi, 700) を呼び出し, 出力側 DCG, output, で出庫指示票または在庫不足票を作成する. この時必要なら在庫データ・ベースを更新する. なお output では出力データ・リストを作成する代わりに直接 write 組込み述語を使用して出力データを印書している. これによってorderの処理が終了し, よって tran の処理が終了し, 次の trans が呼び出される. 残りの入力データが空であるので trans より empty が呼び出され, トークン・リストの解析は終了する. 入力データの解析がすべて終了すれば, 酒倉庫在庫管理プログラムは終了する.

4. 試　行

図-9 のような検査データを用い試行した結果を図-10 に示す. なお試行例で使用している list_stock は, 在庫データ・ベースの内容を印書するデバッキング・エイドである.

```
@TYPE ycontc.m0                    ←入力ファイル名
CONT 200 Asahi  600                ←積荷票例
CONT 200 Kirin  100
CONT 100  Suntory 400
CONT 100  Asahi   200
CONT 400  Sapporo  100
ORDER 1500 Torii Asahi 700         ←注文票例
CONT 500 Ebisu  10
ORDER 1000  NUK  Suntory 400
ORDER 2000 ETL Kirin 200
ORDER 3000 Morisawa Ebisu 10
```

図-9　検査データ

```
| ?- list_stock.

yes
| ?- stock_control('ycontc.m0').
SHIP 1500 Torii
 Container 200 Asahi 600
 Container 100 Asahi 100
SHIP 1000 NUK
 Container 100 Suntory 400
OUT_OF_STOCK 2000 ETL Kirin 200
SHIP 3000 Morisawa
 Container 500 Ebisu 10(Empty)

yes
| ?- list_stock.
Container 200 Kirin 100
Container 400 Sapporo 100
Container 200 Asahi 0
Container 100 Asahi 100
Container 100 Suntory 0

yes
```

図-10　試行例

なお，使用した計算機はDEC 2060，Prologの処理系はDECsystem-10 Prolog[8]である．

5. おわりに

以上のようなプログラムの作成手段は一般に文章処理や事務処理の問題に応用できる．DCGを組み入れたPrologを利用して簡単にプログラムできるので，Rapid-prototypingの一方法としても有効である．

謝辞 本研究は工業技術院電子技術総合研究所ソフトウェア部言語処理研究室において受けた技術指導の一部であります．日頃ご指導頂いています同研究所棟上部長および鳥居室長（現在大阪大学基礎工学部教授），ならびに弊社山崎利治氏に感謝いたします．

参考文献

1) 佐々政孝：コンパイラ生成系，情報処理，Vol. 23, No. 9, pp. 802-817 (1982).
2) ACM SIGPLAN: Proceeding of the SIGPLAN '82, Symposium on Compiler Construction, SIGPLAN Notice, Vol. 17, No. 6 (1982).
3) Jackson, M.A.: Principles of Program Design, Academic Press (1975). 邦訳（鳥居宏次），構造的プログラム設計の原理，日本コンピュータ協会 (1980).
4) Colmerauer, A.: Metamorphosis Grammars, Lecture Notes in Computer Science, No. 63, pp. 133-189, Springer-Verlag (1978).
5) Pereira, F. and Warren, D.: Definite Clause Grammars for Language Analysis, Artif. Intell., Vol. 13, pp. 231-278 (1980).
6) 森沢好臣，鳥居宏次：仕様からPrologプログラムを作成する一手法，情報処理学会第27回（昭和58年後期）全国大会論文集，pp. 507-508 (1983).
7) Torii, K., Morisawa, Y., Sugiyama, Y. and Kasami, T.: Functional Programming and Logical Programming for the Telegram Analysis Problem, 7th International Conference on Software Engineering, pp. 463-472 (1984).
8) Pereira, L., Pereira, F. and Warren, D.: User's Guide to DECsystem-10 Prolog, Div. de Informatica, LNEC, Lisbon and Dept. of AI, University of Edinburgh (1978).

（昭和59年8月27日受付）