ソフトウェア工学 46－2
（1986. 2. 6）

仕様記述言語TELL/NSLにおける仕様記述からプロトタイププログラムへの変換

市川 至　　蓬莱 尚幸　　佐伯 元司　　米崎 直樹　　榎本 肇+

（東京工業大学 工学部）

1.はじめに

自然言語風仕様記述言語TELL/NSL[1] は，自然言語の語彙分割法により自然な形で詳細化が行われるため，人間にとって記述性，了解性に富むという特徴を持っている．さらにその意味は1階述語論理式として形式的に得ることができ，仕様記述自身の無矛盾性やプログラムの正当性といった各種の性質について検証を行える数学的基盤を持っている．

TELL/NSLには，システムの入出力関係を記述する静的記述と，システムの動作を記述する動的記述が用意されている．今回は，TELL/NSLにおける静的記述からプロトタイプとして実行可能なPrologプログラムを，自動的変換により直接的に得て，仕様に対するデータ試験をおこなうプロトタイピング手法について報告する．

変換は，図1のように，TELL/NSLの記述より，その意味となる1階述語論理式を介して，ホーン節形式に変換する．さらに，Prologの実行制御を考慮し，述語の入出力モードを入出力依存グラフを利用して決定し，これにもとずき実行可能なPrologプログラムに変換している．

この変換のように厳密な意味論を持つ仕様記述から，その意味を介して，変換により直接的にプログラム得る手法は，人間によりプログラムを得る手法に比べて，得られたプログラムが仕様を満たしていることの論理的根拠が存在する．

図2は，ページのフォーマット問題について自然言語によるユーザからの要求仕様で，文献[2] より引用した．このような自然言語による仕様記述を，仕様記述者が図3のようなTELL/NSLの記述にする．この記述に対して，変換により得られるPrologプログラムを実行させデータ試験をおこない，仕様記述が仕様記述者の意図したように記述されているかどうかを確かめた．その結果，後に述べるように，一見完全な記述であるように見えるこの仕様が，不明確な記述を含んでいることが判明し，仕様の改善がおこなうことが可能となった．

2.TELL／NSL

TELL/NSLは，限定された自然言語（英語）を基本とし，語彙分割技法[3] と呼ばれる詳細化手法を用いて，問題固有の語句の定義を，自然言語の語句に添ってより原始的な語句へと分解していくことにより進めていく．TELL/NSLではisやnot等の一般の語句はその意味が既に定義済みであるとして利用できる．

TELL/NSLには，システムの入出力関係を記述する静的記述と，システムの動作を記述する動的記述が用意されているが今回は静的記述のみを対象とする．

+ 現在，富士通国際情報科学研究所

2.1.TELL/NSLの語句定義

TELL/NSLの静的記述には，語句の定義方法として，関数定義とクラス定義の2種類があり，それぞれ構文規則に対応して割り当てられた意味規則によって，意味となる1階述語論理式に変換される．

図2をTELL/NSLにより記述した例は図3であり，騎士巡回問題を記述した例は図4である．この中で定義する語句には下線が引かれている．

TELL/NSLでは，共通な性質を持つインスタンスの集まりは普通名詞としてクラス定義により定義される．クラス定義では，定義されるクラス名，クラスの属性を表す語句，クラスの構成子を表す語句，さらにはそのクラスについての構成子や関連する語句も同時に定義する．図3で定義されているクラス名は，file，page，line number である．

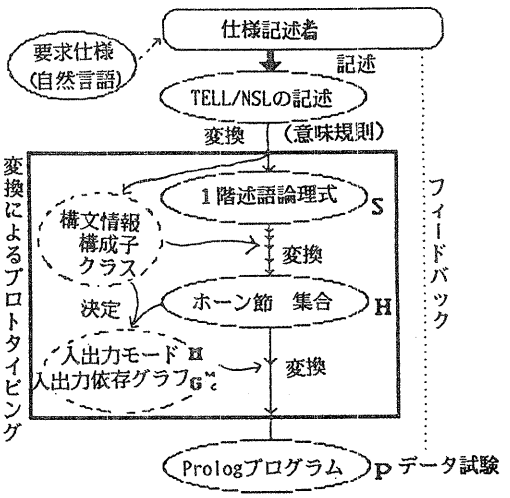


図1．変換によるプロトタイピングの構成

The input to the procedure is a file of program specifications. The object of the procedure is to paginate the input by separating the input into pages. The output is the paginated text in a file ready for printing. Each page must contain some minimum number of lines, denoted MIN. There is also a maximum number of lines, MAX, which is the physical page length. Between the MIN and MAX number of lines, the most important line of text should be chosen to start the next page. Important lines are defined by the following (in order of importance).

1) The first line of the input specification.
2) Those lines which are preceded by blank lines (indicating the beginning of a new specification). The more blank lines preceding a line, the more important that line is.
3) The least indented line (on the assumption that the specifications are written like structured programs with meaningful indenting).

In the case of ties (e.g., two lines each preceded by four blank lines), the line closest to MAX should be chosen for the next page. In case there are no important lines encountered, the page should be broken at MAX lines.

図2．自然言語による要求仕様ーページのフォーマット問題

また，例えば，図３のlineは，文字の列のクラスとして定義されているが，このように直積，直和，集合や列などのよく使用される概念についてはTELL/NSLではあらかじめ用意されており，その記述が利用できる．列に関する記述から，図３のlineの構成子はempty とconcatenation となる．

TELL/NSLでは，クラスのインスタンス間の関係を規定するような普通名詞・形容詞を，

定義文　means that　定義本体　end;

の形式の関数定義で記述する．

図３の最初の関数定義では定義語句はpagenated である．この定義文では，pagenated と入出力を表すT，F，Min，Max の関係を定義しており，例えば，F はfileのクラスを持ちpagenated に対しては前置詞ofを伴って出現することを表している．定義本体は，定義文で定義した入出力の満たすべき条件を場合分けや箇条書きの文を用いて記述する．

図３の中で，関数定義で定義された定義語句は，他にsepa rate a page，break，most important等である．separate a page がpagenated の中にネストして定義されている．このような場合には，ネストの中の定義では外の入出力を陽に定義文で定義しなくとも参照することができる．

今回の発表では，仕様中に現れる語句がすべて仕様記述者により定義され，未定義語句を含まない正しい無矛盾なTELL/NSLの静的記述を対象とする．

```
Pages T is pagenated from file F between minimum line number Min
  and maximum line number Max means that
   case 1) F is empty: T is empty.
   case 2) F is not empty:
        2-1) The result of separating a page from F is
               page P and file F1.
        2-2) Pages T1 is pagenated from F1 between Min and Max.
        2-3) T is the concatenation of P and T1.
  The result of separating a page from file F is page P
    and file F1 means that
     1) Line number N is the most important in F from Min+1 to Max+1.
     2) The result of breaking F at line number N is P and F1.
    The result of breaking file F at line number N is
      file F1 and file F2 means that
       1) F1 is the first half sequence of F with line number N-1.
       2) F2 is the last half sequence of F with line number N.
    end break;
  end separate a page;
  Line number N is the most important in file F from line number I
    to line number J means that
     case 1) I is J: N is J.
     case 2) J is greater than I:
             Let line number M be the most important
               in F from I to J-1,
             case2-1) M is the more important than J in F: N is M.
             case2-2) M is not the more important than J in F: N is J.
    Line number I is the more important line number
      than line number J in file F means that
       Let NI be the number of blank lines in F before I, and
           NJ be the number of blank lines in F before J,
       case 1) NI is greater than NJ .
       case 2) NI is NJ:
            2-1) II is the level of indentation of I th element of F.
            2-2) IJ is the level of indentation of J th element of F.
            2-3) II is greater than IJ .
    end the more important;
    Number N is the number of blank lines in file F
      before line number I means that
       case 1) I is 1: N is 0.
       case 2) I is not 1:
         case 2-1) I - 1 th element of F is not blank line: N is 0.
         case 2-2) I - 1 th element of F is blank line:
              2-2-1) N1 is the number of blank lines in F before I1.
              2-2-2) N is N1 + 1.
    end number of blank lines;
```

2.2.TELL/NSLの意味となる論理式

TELL/NSLで記述された関数的仕様記述は，TELL/NSLの意味規則によってその意味となる１階述語論理式に変換される．TELL/NSLの構文・意味規則の詳細は，ここでは述べないので文献[1,4] を参照されたい．

図５と図６はそれぞれ図３と図４に対して得られる意味となる論理式の一部である．このように，各語句を定義するモジュールの意味は，定義語句に対応する述語，関数を定義する固有の形に制限された論理式（以下，「定義式」と呼ぶ）として得られ，仕様全体の意味はこれらの集合となる．

TELL/NSLの記述からは意味となる論理式の他に，論理式中に現れる関数記号が構成子かどうか，述語がクラスのメンバーシップを表す述語（クラス述語）かどうかといった情報や，クラスの階層関係などの構文情報を得ることができる．

解釈は，構成子のみからなる項を基礎項とし，基礎項を値の定義域とする構造の下で，仕様全体の意味を公理として通常の１階述語論理の解釈を行う．ただし，ｘ＝ｙという同等関係が成立するのは，ｘとｙが同一の形を持つ基礎項に解釈される場合のみであるとする．

複数の基礎項について，反射・対称・推移性が成立する同値関係$\sim_e$（例えば，整数のクラスで，suc を+1，pre を-1とする構成子とする場合，suc(pre(X)) $\sim_e$ Xであるというような関係$\sim_e$）は同等関係＝とは別に仕様記述者が仕様中で定義することになっている．

```
      Number N is the level of indentation of line L means that
         case 1) The first charactor of L is not blank: N is 0.
         case 2) The first charactor of L is blank:
              2-1) N1 is the level of indentation of tail of L.
              2-2) N is N1+1.
       end level of indentation;
  end the most important;
end pagenated;
Number is positive integer.
Line number is positive integer.
Pages is sequence of page.
Page is file.
Line is sequence of charactor associated with tail.
   Line L1 is the tail of line L means that
        1) L is the concatenation of charactor C and line L1.
    end tail;
end line;
File is sequence of line
  associated with first half sequence, last half sequence.
  File F1 is the first half sequence of file F
    with line number N means that
     case 1) N is 0: F1 is empty file.
     case 2) N is not 0:
          2-1) F is concatenation L and F2.
          2-2) F3 is first half sequence of F2 with N -1.
          2-3) F1 is concatenation L and F3.
  end first half sequence;
  File F1 is the last half sequence of file F
    with line number N means that
     case 1) N is 1: F1 is F.
     case 2) N is not 1:
          2-1) F is concatenation L and F2.
          2-2) F1 is last half sequence of F2 with N -1.
  end last half sequence;
end file;
lexicon
Minimum line number := line number.
Maximum line number := line number.
Blank line := empty.
```

図３．TELL/NSLによる仕様記述例

3．変換手法

変換は，先の図1に示した構成をとる．まず，TELL/NSLの記述より意味規則を用いて，TELL/NSLの意味として得られる論理式（定義式）の集合Sを得る．次に，Sをホーン節の集合Hに変換する．さらにPrologの実行制御を考慮し，述語の入出力モードを入出力依存グラフを利用して決定する．これにもとずきリテラルや節の並べ換えを行い，Hから実行可能なPrologプログラムPに変換する．

これらの変換は，多段的に適応されるいくつかの部分変換からなり，各部分変換規則は書き換え規則の集まりであり，

```
Trace T is a knight's tour means that
  1) T is a knight's unique trace.
  2) Every field is visited in T.
end knight's tour;

Trace T is a knight's unique trace means that
  case1) T is empty sequence.
  case2) T is not an empty sequence:
         let T be the concatenation of field F and
             knight's unique trace T1 ,
    case2-1) T1 is empty sequence:
    case2-2) otherwise:
        2-2-1) F is not the member of T1.
        2-2-2)let T1 be the concatenation of field F1 and
             knight's unique trace T2 ,
          F1 and F are in knight's moving relation.
end knight's unique trace;

Field F1 and field F2 are in knight's moving relation means that
    case1) F1 is left to F2:
        case1-1) F3 is below to F1: F2 is below to F3.
        case1-2) F3 is below to F2: F1 is below to F3.
    case2) F2 is left to F1:
        case2-1) F3 is below to F1: F2 is below to F3.
        case2-2) F3 is below to F2: F1 is below to F3.
    case3) F1 is below to F2:
        case3-1) F3 is left to F1: F2 is left to F3.
        case3-2) F3 is left to F2: F1 is left to F3.
    case4) F2 is below to F1:
        case4-1) F3 is left to F1: F2 is left to F3.
        case4-2) F3 is left to F2: F1 is left to F3.
end in knight's moving relation;

Field F is visited in trace T means that
  1) F is a member of T.
end visited;

Trace is associated with empty sequence, concatenation, and member
construction
  1) empty sequence is trace,
  2) the result of the concatination of trace T
       and field F is trace.
    Field F is member of trace T means that
        case 1) T is the concatenation of T1 and field E
            case 1-1) E is F.
            case 1-2) F is the member of T1.
    end member.

    Field is < X: Row, Y: Column >.

    Field F1 is left to field F2 means that
        case1) /x(F1)/ is A: /x(F2)/ is B.
        case2) /x(F1)/ is B: /x(F2)/ is C.
        case3) /x(F1)/ is C: /x(F2)/ is D.
        case4) /x(F1)/ is D: /x(F2)/ is E.
        case5) /x(F1)/ is E: /x(F2)/ is F.
        case6) /x(F1)/ is F: /x(F2)/ is G.
        case7) /x(F1)/ is G: /x(F2)/ is H.
    end left to.

    Field F1 is below to field F2 means that
        case1) /y(F2)/ is I: /y(F1)/ is II.
        case2) /y(F2)/ is II: /y(F1)/ is III.
        case3) /y(F2)/ is III: /y(F1)/ is IV.
        case4) /y(F2)/ is IV: /y(F1)/ is V.
        case5) /y(F2)/ is V: /y(F1)/ is VI.
        case6) /y(F2)/ is VI: /y(F1)/ is VII.
        case7) /y(F2)/ is VII: /y(F1)/ is VIII.
    end below to.
end trace.

Column is I,II,III,IV,V,VI,VII or VIII.
end column.

Row is A,B,C,D,E,F,G or H.
end row.
```

図4．TELL/NSLによる仕様記述例－騎士巡回問題

```
∀T ∀F ∀Min ∀Max[pagenated(T,F,Min,Max)↔
    pages(T)∧file(F)∧line_number(Min)∧line_number(Max)∧
    ((F=empty∧T=empty)∨
     (¬F=empty∧∃F1[file(F1)∧∃P[page(P)∧∃T1[pages(T1)∧
           separate_a_page(P,F1,F,Min,Max)∧
           pagenated(T1,F1,Min,Max)∧T=concatenation(P,T1)]]]))]

∀P ∀F1 ∀F[separate_a_page(P,F1,F,Min,Max)↔
    page(P)∧file(F1)∧file(F)∧line_number(Min)∧line_number(Max)∧
    ∃N[line_number(N)∧most_important(N,F,Min+1,Max+1)∧break(P,F1,F,N)]]

∀F1 ∀F2 ∀F ∀N[break(F1,F2,F,N)↔
    file(F1)∧file(F2)∧file(F)∧line_number(N)∧
    first_half_sequence(F1,F,N-1)∧last_half_sequence(F2,F,N)]

∀N ∀F ∀I ∀J[most_important(N,F,I,J)↔
    line_number(N)∧file(F)∧line_number(I)∧line_number(J)∧
    ((I=J∧N=J)∨(J>I∧∃M[number(M)∧
          most_important(M,F,I,J-1)∧(more_important(M,J,F)∧N=M∨
          ¬more_important(M,J,F)∧N=J))])]

∀I ∀J ∀F[more_important(I,J,F)↔
    line_number(I)∧line_number(J)∧file(F)∧
    ∃NI[number(NI)∧NI=number_of_blank_lines(F,I)∧
        ∃NJ[number(NJ)∧NJ=number_of_blank_lines(F,J)∧
            (NI>NJ∨(NI=NJ∧∃II[number(II)∧
                II=level_of_indentation(element(F,I))∧
                ∃IJ[number(IJ)∧IJ=level_of_indentation(element(F,J))∧
                  II>IJ]))]]]

∀N ∀F ∀I[N=number_of_blank_lines(F,I)↔
    number(N)∧file(F)∧number(I)∧
    ((I=1∧N=0)∨(¬empty=element(F,I-1)∧N=0)∨
     (empty=element(F,I-1)∧∃N1[number(N1)∧
        N1=number_of_blank_lines(F,I-1)∧N=N1+1]))]

∀V0[number(V0)↔positive_integer(V0)]

∀V0[line_number(V0)↔positive_integer(V0)]

∀V0[pages(V0)↔
    V0=empty∨∃V1[page(V1)∧∃V2[pages(V2)∧V0=concatenation(V1,V2)]]]

∀V0[page(V0)↔file(V0)]

∀V0[line(V0)↔
    V0=empty∨∃V1[charactor(V1)∧∃V2[line(V2)∧V0=concatenation(V1,V2)]]]

∀V0 ∀V1[V0=first_charactor(L)↔
   charactor(V0)∧line(V1)∧∃V2[line(V2)∧V1=concatenation(V0,V2)]]

∀L1 ∀L[L1=tail(L)↔
    line(L1)∧line(L)∧∃C[charactor(C)∧L=concatenation(C,L1)]]

∀V0 ∀V1[V0=first_cahractor(V1)↔
    charactor(V0)∧line(V1)∧∃V2[line(V2)∧V1=concatenation(V0,V2)]]

    ∀V0[charactor(V0)↔V0=' 'V ... VV0='z']
```

図5．図3の意味となる論理式（部分）

```
∀T[knight_s_tour(T)↔
   trace(T)∧
       knight_s_unique_trace(T)∧∀V0[field(V0)→visited(V0,T)]]

∀T[knight_s_unique_trace(T)↔
    trace(T)∧(T=empty_sequence∨
          (¬T=empty_sequence∧
            ∃F ∃T1[field(F)∧knight_s_unique_trace(T1)∧
             T=concatenation(F,T1)∧(T1=empty_sequence∨
               (¬T1=empty_sequence∧
                 ∃F1 ∃T2[field(F1)∧knight_s_unique_trace(T2)∧
                  T1=concatenation(F1,T2)∧
                  in_knight_s_moving_relation(F,F1)])])))]

∀F1 ∀F2[in_knight_s_moving_relation(F1,F2)↔
    field(F1)∧field(F2)∧
        ((left(F1,F2)∧
            (∃F3[field(F3)∧below(F3,F1)∧below(F2,F3)]∨
             ∃F3[field(F3)∧below(F3,F2)∧below(F1,F3)]))∨
         (left(F2,F1)∧
            (∃F3[field(F3)∧below(F3,F1)∧below(F2,F3)]∨
             ∃F3[field(F3)∧below(F3,F2)∧below(F1,F3)]))∨
         (below(F1,F2)∧
            (∃F3[field(F3)∧left(F3,F1)∧left(F2,F3)]∨
             ∃F3[field(F3)∧left(F3,F2)∧left(F1,F3)]))∨
         (below(F2,F1)∧
            (∃F3[field(F3)∧left(F3,F1)∧left(F2,F3)]∨
             ∃F3[field(F3)∧left(F3,F2)∧left(F1,F3)])))]

∀F ∀T[visited(F,T)↔field(F)∧trace(T)∧member(F,T)]

∀V0[trace(V0)↔V0=empty_sequence∨
    ∃V1 ∃V2[field(V1)∧trace(V2)∧V0=concatenation(V1,V2)]]

∀F ∀T[member(F,T)↔field(F)∧trace(T)∧
        ∃E ∃T1[field(E)∧trace(T1)∧T=concatenation(E,T1)∧
          (E=F∨member(F,T1))]]

∀V0[field(V0)↔∃V1∃V2[row(V1)∧column(V2)∧V0=tuple(V1,V2)]]

∀V0 ∀V1[V1=x(V0)↔field(V0)∧row(V1)∧∃V2[column(V2)∧V0=tuple(V1,V2)]]

∀V0 ∀V1[V2=y(V0)↔field(V0)∧column(V2)∧∃V2[row(V1)∧V0=tuple(V1,V2)]]

∀F1 ∀F2[left(F1,F2)↔
    field(F1)∧field(F2)∧((x(F1)=A∧x(F2)=B)∨
          (x(F1)=B∧x(F2)=C)∨(x(F1)=C∧x(F2)=D)∨
          (x(F1)=D∧x(F2)=E)∨(x(F1)=E∧x(F2)=F)∨
          (x(F1)=F∧x(F2)=G)∨(x(F1)=G∧x(F2)=H))]

∀F1 ∀F2[below(F1,F2)↔
    field(F1)∧field(F2)∧((y(F2)=I∧y(F1)=II)∨
          (y(F2)=II∧y(F1)=III)∨(y(F2)=III∧y(F1)=IV)∨
          (y(F2)=IV∧y(F1)=V)∨(y(F2)=V∧y(F1)=VI)∨
          (y(F2)=VI∧y(F1)=VII)∨(y(F2)=VII∧y(F1)=VIII))]

∀V0[row(V0)↔
    V0=A∨V0=B∨V0=C∨V0=D∨V0=E∨V0=F∨V0=G∨V0=H]

∀V0[column(V0)↔
    V0=I∨V0=II∨V0=III∨V0=IV∨V0=V∨V0=VI∨V0=VII∨V0=VIII]
```

図6．図4の意味となる論理式（部分）

部分変換は適応できる書き換え規則がなくなると変換を終了する．すべての部分変換は閉世界仮説の下で健全性が保証されており，すなわち，どの仕様に対してもこの手法で得られるプログラムが解を得ればその解は仕様を満たしていることが保証されている．

3.1.関数の述語化

TELL/NSLの意味となる論理式は，構成子以外の関数が項を構成することを認めている．Prologに変換する為に，項を構成する関数は定義されている構成子のみとし，かつ，それら以外の n引数関数f は n+1引数の述語f'を用いて書き換える．この変換が健全であるためには，述語形式にする関数f が t=f(ξ) ∧s=f(ξ) →t=s という関数の一値性に矛盾しない関数であることが必要である．

3.2.ホーン節への変換

次に，定義ごとの論理式を，そこで定義される定義語句名をそのまま述語記号として，これを頭部とするホーン節に変換する．

［変換1 ］定義式の解釈変更：
定義式の”←→”を”←”に書き換える．
∀x1...xn [R(x1,・・,xn)←→Φ]]が定義式ならば，
∀x1...xn [R(x1,・・,xn)←Φ]] に書きかえる．

なお以降，本体とは∀x1...xn [R(x1,・・,xn)←Φ］のΦをさすものとし，本体中とは，このΦの部分式をさすものとする．

［変換2 ］本体中の含意及び全称限量子の除去：

［変換3 ］本体中の否定の内部移動と二重否定の除去:
この変換では，”～∃”が”∀～”になることはない．

ここまでの変換を行った結果，本体中では，限量は，存在限量子によってのみおこなわれ，接続詞は∧∨のみで，否定記号～は原子式または存在限量された式にしか用いられていない．

［変換4 ］本体中の存在否定の除去：
本体中の～∃x[Φ］（存在否定）を新しい述語記号πを用いて，～π（x1,・・・・,xn）に書き換え，新しく
∀x1 ...∀xn[π（x1,・・・・,xn）←∃x[Φ]]を加える．ここで，x1,・・・,xn はΦでfreeに出現する x以外の変数とする．

［変換5 ］全称・存在限量の除去：
例：∀x1...xn[R(x1,...,xn)←
∃y1...ym[Q(y1,...,ym,x1,...,xn)]]
=> R(x1,...,xn)←Q(z1,...,zm,x1,...,xn)

［変換6 ］選言標準形化：
ただし本体の否定された式を頭部に移動しない．

［変換7 ］同等関係についての簡略化：

図５の定義式集合から，ここまでの変換で得られたホーン節集合を図７に示す．

```
pagenated(empty.empty,Min,Max) ←
        file(empty)∧pages(empty)∧line_number(Min)∧line_number(Max)
pagenated(concatenation(V02,V03),F,Min,Max) ←
        file(F)∧pages(concatenation(V02,T1))∧
        line_number(Min)∧line_number(Max)∧~(F=empty)∧
        file(V01)∧page(V02)∧pages(V03)∧
        separate_a_page(V02,V01,F,Min,Max)∧
        pagenated(V03,V01,Min,Max)
separate_a_page(P,F1,F,Min,Max) ←
        page(P)∧file(F1)∧file(F)∧line_number(Min)∧line_number(Max)∧
        line_number(V11)∧~'(V12,Min,1)∧~'(V13,Max,1)∧
        most_important(V11,F,V12,V13)∧break(P,F1,F,V11)
break(F1,F2,F,N) ←
        page(F1)∧file(F2)∧file(F)∧line_number(N)∧
        ~'(V21,N,1)∧first_half_sequence(F1,F,V21)∧
        last_half_sequence(F2,F,N)
most_important(V31,F,V31,V31) ← line_number(V31)∧file(F)
most_important(V42,F,I,J) ←
        line_number(N)∧file(F)∧line_number(I)∧line_number(J)∧
        >(J,I)∧~'(V41,J,1)∧line_number(V42)∧
        most_important(V42,F,I,V41)∧more_important(V42,J,F)
most_important(J,F,I,J) ←
        line_number(N)∧file(F)∧line_number(I)∧line_number(J)∧
        >(J,I)∧~'(V41,J,1)∧line_number(V42)∧
        most_important(V42,F,I,V41)∧~(more_important(V42,J,F))
more_important(I,J,F) ←
        line_number(I)∧line_number(J)∧file(F)∧
        line_number(V51)∧number_of_blank_lines'(V51,F,I)∧
        line_number(V52)∧number_of_blank_lines'(V52,F,J)∧
        >(V51,V52)
more_important(I,J,F) ←
        line_number(I)∧line_number(J)∧file(F)∧
        line_number(V51)∧number_of_blank_lines'(V51,F,I)∧
        number_of_blank_lines'(V51,F,J)∧line_number(V52)∧
        element'(V53,F,I)∧level_of_indentation'(V52,V53)∧
        line_number(V54)∧element'(V55,F,J)∧
        level_of_indentation'(V54,V55)∧>(V52,V54)
number(V0) ← positive_integer(V0)
line_number(V0) ← positive_integer(V0)
pages(empty)
pages(concatenation(V81,V82)) ← page(V81)∧pages(V82)
page(V0) ← file(V0)
line(empty)
line(concatenation(V91,V92)) ← charactor(V91)∧line(V92)
first_char'(V101,concatenation(V101,V102)) ←
        charactor(V101)∧line(concatenation(V101,V102))∧line(V102)
tail'(V111,concatenation(V112,V111)) ←
        line(V111)∧line(concatenation(V112,V111))∧charactor(V112)
charactor(' ')
charactor('z')
```

図７．図５から得られるホーン節集合（部分）

4.Prologプログラムへの変換

以上の変換の結果，ホーン節の集合が得られる．これをまず，Prologプログラムの記法に変換し，次に，各述語の入出力を決定し，それによりリテラルや節の並べ換えをおこない，実行可能なPrologプログラムを得る．

4.1.プログラムの記法への変換と条件

まず，ホーン節をPrologプログラムの記法に変換する．本論文では，Prologとして，DEC10-Prolog，C-Prolog[5] を対象とする．ここで，否定（～）に対しては，Prologのメタ述語 notを用いることにする．

この変換により得られたnot を含むPrologプログラム（節，リテラルの順序はまだ未定）の実行により求まる解が，常に元の仕様を満たす為には，

［条件4.1 ］
①not の中の変数が必ず基礎項に代入された後で実行され，
②実行動作が有限で停止する．

という２つの条件を満たす必要がある．

この為，本手法では，以上の２条件を満たし，かつ，なるべく多くの解が求まるように，リテラルや節の並べ換えの変換を行うことにする．こうした並べ換え自体はプログラムの実行により求まる解の集合を変化させるが，条件4.1 を満たしていれば，Prologの実行により得られた解集合はすべて，元の仕様を満たす解集合の部分集合であることは保証されている．

4.2.入出力の決定

条件4.1 を満たしてリテラルや節の並べ換えを行う為に，まず，変数の代入状態などの動作解析が容易になるように，Prologの動作を制限した次の仮定をおこなう．

［仮定4.2 ］関数的動作仮定：
各リテラルの実行が成功した場合には，各引数は必ず完全に基礎項に代入されているように動作する．

この仮定は，実行が成功する場合には，変数を含む項が変数に代入されることがないことを意味している．

実は，今までの変換でTELL/NSLの記述から得られるPrologプログラム（節，リテラルの順序はまだ未定）では，節の実行が成功した場合には，その引数は必ず基礎項に代入されるという性質があるので，仮定4.2 は自然な仮定である．

この仮定の下で，変数がどこで代入されるかを決定できるように，入出力モードを設定する．これにより，not に対して常に代入された後で実行されるような節の本体のリテラルの順序を決定することが可能になる．

［定義4.3 ］入出力モード
入出力モードは，プログラム中の各述語記号p について，その全ての出現p(x1,･･･,xn)で，xiが入力か出力のいずれかを割り当てたものである(1≦i ≦n)．ここで，入力，出力は，モード値で，以下のように定義する．

・入力：その引数が，基礎項に代入がなされて評価しなければならないことを表す．

・出力：その引数には，そのような制限がないことを表す．

関数的動作仮定の下では，出力は，評価が終われば完全に代入されている場合であり，この手法では出力が衝突することも認められる．

モードの決定やリテラルの順序決定を容易にするために，ベトリネット[6] 的な２分有効グラフを定義する．

[定義4.4]入出力依存グラフ：
節Ｃに対する入出力モードＭの下での入出力依存グラフ
$G^{M}_{C}=<N_C, A^{M}_{C}>$を次のように定義する．

Ｃが Bo:- B1,...,Bn. （n≧0）であるとすると，

N_Cはノードの集合で，次の２種のノード集合からなる．
L_C：リテラルの集合，$L_C=\{Bo,B1,...,Bn\}$
V_C：節中に出現するすべての変数の集合．
$N_C=L \cup V$，$L \cap V=\phi$　である．

A^{M}_{C}は有効アークの集合で，$<a, b>\in A^{M}_{C}$は，ａからｂへのアークを表す．A^{M}_{C}は以下の条件を満たす最小集合として定義される．

①Boが$P(t1,...,tk)$で，モード値が入力のtiにx が出現するならば，$<Bo\ x>\in A^{M}_{C}$，$(1\leq i\leq k)$

②Boが$P(t1,...,tk)$で，モード値が出力のtiにx が出現し，かつ，$<Bo,x>\notin A^{M}_{C}$であるならば，$<Bo,D>\in A^{M}_{C}$，$(1\leq i\leq k)$

③Bjがnot(P(t1,...,tk)) で，tiにx が出現するならば，$<x,Bj>\in A^{M}_{C}$，$(1\leq j\leq n, 1\leq i\leq k)$

④Bjが$P(t1,...,tk)$で，モード値が入力のtiにx が出現するならば，$<x,Bj>\in A^{M}_{C}$，$(1\leq j\leq n, 1\leq i\leq k)$

⑤Bjが$P(t1,...,tk)$で，モード値が出力のtiにx が出現し，かつ，$<x,Bj>\notin A^{M}_{C}$であるならば，$<Bj,x>\in A^{M}_{C}$．$(1\leq j\leq n, 1\leq i\leq k)$

これは，リテラルに入るアークの元にある変数がすべて代入されるとそのリテラルの評価が可能となり，関数的動作仮定の下で評価が終わるとそのリテラルから出たアークの先の変数は全て代入されることを表すためのグラフである．変数の出現する項のモード値により構成されるが，頭部のリテラルのみはモード値を反転して用いる．

4.2.1.入出モードの候補の選択条件

まずあるモードＭを仮定し，以下に述べる条件が満たされなければ他のモードが選ばれる．

①システムが用意している述語記号については，そのモード値は固定されているので，Ｍはこれを満たさねばならない．

②クラスの定義が再帰によりおこなわれるクラスは再帰的なクラスとし，再帰的なクラスを定義に利用するクラスも再帰的なクラスとする．

再帰的なクラスのクラス述語記号の引数のモード値は常に入力に，再帰的でないクラスのクラス述語記号の引数では出力に固定する．Ｍはこれを満たさねばならない．例えば，fileは再帰的なクラスなので，そのモード値は入力とする．

これは，再帰的なクラスのクラス述語が未代入変数を含む引数で実行された場合に，バックトラックが生じると実行が無限の再帰に陥る可能性があるためである．

③Ｍは，Prologプログラム中の全ての節Ｃに対して，次の代入に関する条件4.4 が成立しなければならない．

[条件4.5]入出力モードの代入に関する条件：
節Ｃに対するＭの下での入出力依存グラフ$G^{M}_{C}=<N,A>$において，Ｃに出現する変数x はすべて，$<B, x>\in A$となるリテラルＢが存在する．

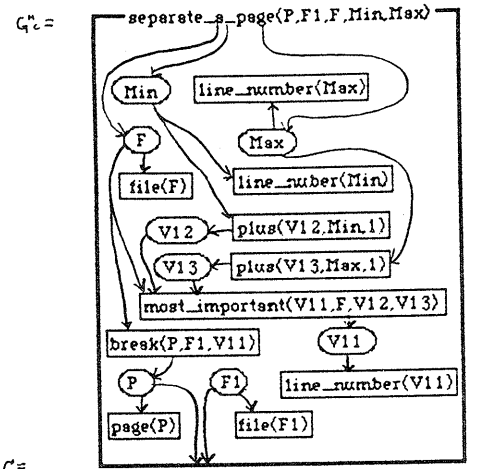
これは，各節のどの変数も，リテラルの順序とは関係なくその節の実行が終了する時には，必ず完全に代入されることを表している．なぜならば，関数的動作仮定の下で，変数に入るアークに出力するいずれかのリテラルによりその変数が代入されることは明らかであるので，入るアークのない変数は，その節の実行によって代入がおこなわれないことを表している．このような変数は関数的動作仮定の下では認められないからである．

以上の３つの条件を満たす入出力モードは，複数個存在するので，候補を次の２つの条件で絞り込み決定する．

④人間が仕様記述時にどの引数を主語にしたかという情報から，仕様記述の最上位のシステムを定義している定義語句に対応する述語（図３ではpagenated 図４ではknight_s_tour）についてそのモード値を設定する．というのは，このモード値は，得られたプログラムをどのように用いるかということに関係しているからである．

⑤ある１つの述語のある引数の位置のモード値だけが入力と出力で異なるようなモードの候補が存在する場合は，出力の方の候補を代表として残す．というのは，出力は代入されている場合を含み，すなわち，入力の場合を含み，かつ，以上の条件がそのモード値に依存していないからである．

図８に図７に対して決定した入出力モードと，それに対する入出力依存グラフの例を示す．図では，入出力モード値の入力を＋で出力を－で表しており，入出力依存グラフは，丸が変数のノードを四角がリテラルのノードを表し，頭部のリテラルは外の大きな四角であり，矢印で有効アークを表している．




```
C=
separate_a_page(P,F1,F,Min,Max) ←
        page(P)∧file(F1)∧file(F)∧line_number(Min)∧line_number(Max)∧
        line_number(V11)∧'(V12,Min,1)∧'(V13,Max,1)∧
        most_important(V11,F,V12,V13)∧break(P,F1,F,V11)

M=
pagenated(-,+,+,+)                  separate_a_page(-,-,+,+,+)
break(-,-,+,+)                      most_important(-,+,+,+)
more_important(+,+,+)               number_of_blank_lines(-,+,+)
level_of_indentation(-,+)           number(+)
line_number(+)                      pages(+)
page(+) line(+)                     first_char(-,+)
tail(-,+)                           file(+)
seq_lement(-,+,+)                   first_half_sequence(-,+,+)
last_half_sequence(-,+,+)           charactor(-)
posiitive_integer(+)                add(-,+,+)
sub(-,-,+)                          greater(+,+)
```

図８．入出力モードと入出力依存グラフの例

4.3.リテラルの順序

通常のPrologでは，節中のリテラルについては左から右の順で実行を行うので，正しく動作するようにリテラルを並べ換える．この並び換えは，まず，4.2 節で決定した入出力モードMを用いて，各節Cごとに次の条件を満たす本体のリテラルの順序を求める．もしある節で順序が求まらないようなモードは，破棄され，別のモードが選ばれる．

[条件4.6]
節Cに対するMの下での入出力依存グラフを$G^{m}_{c}=\langle N,A\rangle$とする．Cが Bo :- B1,...,Bn. ($n\geq 0$) であるとすると，CのモードMでの本体の順序O^{m}_{c}は，次を満たさねばならない．

本体の全てのリテラルBi ($1\leq i\leq n$)で，$\langle x,Bi\rangle\in A$ となる全ての変数x は，頭部のリテラルBoで$\langle Bo,x\rangle\in A$ であるか，または，$\langle Bj,x\rangle\in A$ となりかつ順序O^{m}_{c}でBiより左となるリテラルBj ($1\leq j\leq n$)が存在しなければならない．

この条件は，各節のリテラルのモード値が入力の項に現れるどの変数についても，代入されてから実行されるような順序でなければならないことを示している．

全ての節Cで条件4.6 を満たす順序O^{m}_{c}でリテラルの実行が行われれば，not の中の項は常に基礎項で代入された後に実行される．

１つのモードMにもこの条件を満たす順序O^{m}_{c}が複数存在しうるので，さらに，求まった順序の中からなるべく解が求まりかつ停止するような順序を経験則に基づき決定し，その順序でリテラルを並べ換える．

4.4.節の順序決定

通常のPrologでは，同一の述語名を頭部に持つ節については，上から下の順で実行を行う．リテラルの並べ換えが行われていれるPrologプログラムは，節の順序に関係なく，not は全て代入されてから実行されているが，節の順序によっては停止しなくなる場合がある．

そこで，入出力依存グラフを利用して経験則に基づき，なるべく解が求まりかつなるべく停止するような順序に節の並べ換えを行う．

図３，４のTELL/NSLの記述に対して今までの変換を行った結果，得られるPrologプログラムの一部をそれぞれ図９，10に示す．図９では，このプログラムを実行させて仕様のデータ試験を行うのに，concatenation(A,B)といった記法では試験値の作成に不便なので，便法としてC-Prologのリストの記法[]，[A | B]などを用いてプログラムを書き換えてある．

```
pagenated([],[],Min,Max) :-
        line_number(Min),line_number(Max),file(empty),pages(empty).
pagenated([V02|V03],F,Min,Max) :-
        line_number(Min),line_number(Max),file(F),pages([V02|V03]),
        not(F=[]),separate_a_page(V02,V01,F,Min,Max),
        file(V01),page(V02),pagenated(V03,V01,Min,Max),pages(V03).

separate_a_page(P,F1,F,Min,Max) :-
        file(F),line_number(Min),line_number(Min),
        add(V12,Min,I),add(V13,Max,1),most_important(V11,F,V12,V13),
        break(P,F1,F,V11),pages(P),file(F1).

break(F1,F2,F,N) :-
        file(F),line_number(N),sub(V21,N,1),
        first_half_sequence(F1,F,V21),page(F1),
        last_half_sequence(F2,F,N),file(F2).
most_important(V31,F,V31,V31):- line_number(V31),file(F).
most_important(V42,F,I,J):-
        line_number(I),line_number(J),file(F),greater(J,I),
        sub(V41,J,1),most_important(V42,F,I,V41),line_number(V42),
        more_important(V42,J,F).
most_important(J,F,I,J):-
        line_number(I),line_number(J),file(F),greater(J,I),
        sub(V41,J,1),most_important(V42,F,I,V41),line_number(V42),
        not(more_important(V42,J,F)).
more_important(I,J,F):-
        line_number(I),line_number(J),file(F),
        number_of_blank_lines(V51,F,I),line_number(V51),
        number_of_blank_lines(V52,F,J),line_number(V52),
        greater(V51,V52).
more_important(I,J,F):-
        line_number(I),line_number(J),file(F),
        number_of_blank_lines(V51,F,I),line_number(V51),
        number_of_blank_lines(V51,F,J),seq_element(V53,F,I),
        level_of_indentation(V52,V53),line_number(V52),
        seq_element(V55,F,J),level_of_indentation(V54,V55),
        greater(V52,V54).
line_number(V0) :- positive_integer(V0).
pages([]).
pages([V81|V82]) :-page(V81),pages(V82).
page(V0) :- file(V0).
line([]).
line([V91|V92]) :- charactor(V91),line(V92).
tail(V111,[V112|V111]):- line(V111),line([V112|V111]),char(V112).
first_char(V101,[V101|V102]):- char(V101),line([V101|V102]),line(V102).
file([]).
file([V121|V122]) :- line(V121),file(V122).
seq_element(V131,[V131|V135],1).
seq_element(V131,[V134|V135],V133) :-
        sub(V136,V133,1), seq_element(V131,V135,V136).
first_half_sequence([],F,0).
first_half_sequence([V141|V143],[V141|V142],N) :-
        sub(V144,N,1), first_half_sequence(V143,V142,V144).
last_half_sequence(F1,F1,1).
last_half_sequence(F1,[V151|V152],N) :-
        sub(V153,N,1), last_half_sequence(F1,V152,V153).
```

図9．図3から得られたPrologプログラム（部分）

```
knight_s_tour(T) :- knight_s_unique_trace(T), not(p0(T)).
p0(T):- field(V0), not( visited(V0,T) ).
knight_s_unique_trace(empty_sequence).
knight_s_unique_trace(concatenation(F,empty_sequence)) :- field(F).
knight_s_unique_trace(concatenation(F,concatenation(F1,T2))) :-
        in_knight_s_moving_relation(F,F1),
        knight_s_unique_trace(T2).
in_knight_s_moving_relation(F1,F2) :- left(F1,F2),below(F3,F1),below(F2,F3).
in_knight_s_moving_relation(F1,F2) :- left(F1,F2),below(F3,F2),below(F1,F3).
in_knight_s_moving_relation(F1,F2) :- left(F2,F1),below(F3,F1),below(F2,F3).
in_knight_s_moving_relation(F1,F2) :- left(F2,F1),below(F3,F2),below(F1,F3).
in_knight_s_moving_relation(F1,F2) :- below(F1,F2),left(F3,F1),left(F2,F3).
in_knight_s_moving_relation(F1,F2) :- below(F1,F2),left(F3,F2),left(F1,F3).
in_knight_s_moving_relation(F1,F2) :- below(F2,F1),left(F3,F1),left(F2,F3).
in_knight_s_moving_relation(F1,F2) :- below(F2,F1),left(F3,F2),left(F1,F3).
visited(F,T) :- member(F,T).
member(F,concatenation(F,T1)).
member(F,concatenation(E,T1)) :- member(F,T1).
field(tuple(V1,V2)) :- row(V1),column(V2).
x(V1,tuple(V1,V2)) :- row(V1),column(V2).
y(V2,tuple(V1,V2)) :- column(V2),row(V1).
left(F1,F2) :- x(a,F1),x(b,F2).
left(F1,F2) :- x(b,F1),x(c,F2).
left(F1,F2) :- x(c,F1),x(d,F2).
left(F1,F2) :- x(d,F1),x(e,F2).
left(F1,F2) :- x(e,F1),x(f,F2).
left(F1,F2) :- x(f,F1),x(g,F2).
left(F1,F2) :- x(g,F1),x(h,F2).
below(F1,F2) :- y(i,F2),y(ii,F1).
below(F1,F2) :- y(ii,F2),y(iii,F1).
below(F1,F2) :- y(iii,F2),y(iv,F1).
below(F1,F2) :- y(iv,F2),y(v,F1).
below(F1,F2) :- y(v,F2),y(vi,F1).
below(F1,F2) :- y(vi,F2),y(vii,F1).
below(F1,F2) :- y(vii,F2),y(viii,F1).
row(a).
row(b).
row(c).
row(d).
row(e).
row(f).
row(g).
row(h).
column(i).
column(ii).
column(iii).
column(iv).
column(v).
column(vi).
column(vii).
column(viii).
```

図10．図4から得られたPrologプログラム

5.プロトタイプによる仕様のデータ試験

図2の自然言語で記述された要求仕様に対するTELL/NSLによる記述は図3で，この手法により最終的に得られたプログラムは図9である．

この図9のプログラムをファイル，最大行数，最小行数を与えて実行してデータ試験をすることにする．ここで，get_fileとprint_out という述語を，ファイルの入力と結果の印刷の為に用意しておく．

図11は今回の試験のためのファイル入力を印刷した例であり，ファイルは文字列の列となっており，""は空白行を表している．

```
|?- get_file(X),printer(X,"INPUT").

INPUT =[
        "main()",
        "{",
        "    int i;",
        "",
        "    for (i = 0; i<10; ++i)",
        "        printf(i,power(2,i),power(-3,i));",
        "}",
        "",
        "",
        "",
        "power(x,n)",
        "int x, n;",
        "{",
        "    int i, p;",
        "",
        "",
        "    p=1;",
        "    for (i=1;i<=n;++i)",
        "        p=p*x;",
        ",    return(p)",
        "}",
        "",
        "",
        ""
]
```

図11．試験のためのファイル入力

図12は，このファイル入力に対して，最小行数5，最大行数8と最小行数5，最大行数7の2つの場合に実行した結果である．

前者は成功し，結果の出力は文字列の列の列となっており，文字列の列が1ページを表している．これに対して後者は失敗している．

```
|?- Min=5,Max=8,get_file(Input),pagenated(Output,Input,Min,Max),printer(Output).

OUTPUT=[
        [
                "main()",
                "{",
                "    int i;",
                "",
                "    for (i = 0; i<10; ++i)",
                "        printf(i,power(2,i),power(-3,i));",
                "}",
                ""
        ],[
                "",
                "",
                "power(x,n)",
                "int x, n;",
                "{",
                "    int i, p;",
                "",
                ""
        ],[
                "    p=1;",
                "    for (i=1;i<=n;++i)",
                "        p=p*x;",
                "    return(p)",
                "}",
                "",
                "",
                ""
        ]
]

|?- Min=5,Max=7,get_file(Input),pagenated(Output,Input,Min,Max),printer(Output).

no.
```

図12．実行結果

同じファイル入力に対して最小／最大行数を変化させて実行させてみると，ファイルから１ページずつ切りだしていき，最後に行が残らなかった場合のみ成功していることが判明した．このことは，図２の要求仕様が完全でなく，記述が不足していおり，仕様記述者がこの点を見逃したと考えることができる．確かに，ファイルの行数が，最大行数より少ない場合についての記述が述べられていない．

そこで，TELL/NSLの記述でseparate a page にファイルの行数についての条件を追加し，変換するとseparate_a_pageの部分が図13のように変化したPrologプログラムを得た．

```
separate_a_page(F,[], F, Min, Max) :-
        file(F),line_number(Min),line_number(Max),pages([]),
        seq_length(V12,F),not(V12 > Max).

separate_a_page(P,F1, F, Min, Max) :-
        file(F),line_number(Min),line_number(Max),
        seq_length(V12,F),V12 > Max,
        add(V13,Min,1),add(V14,Max,1),most_important(V11,F,V13,V14),
        break(P,F1,F,V11),pages(P),file(F1).
```

図13．プログラムの変更部分

これを先のファイル入力に対して実行してみたところ，最小行数５，最大行数８および最小行数５，最大行数７いずれの場合もページングが成功した．前者は図12と同じ結果を得たので，後者の結果を図14に示す．

```
|?- Min=5,Max=7,get_file(Input),pagenated(Output,Input,Min,Max),printer(Output).

OUTPUT=[
        [
                "main()",
                "{",
                "    int i;",
                "",
                "    for (i = 0; i<10; ++i)"
        ],[
                "        printf(i,power(2,i),power(-3,i));",
                "}",
                "",
                "",
                ""
        ],[
                "power(x,n)",
                "int x, n;",
                "{",
                "    int i, p;",
                "",
                ""
        ],[
                "    p=1;",
                "    for (i=1;i<=n;++i)",
                "        p=p*x;",
                "    return(p)",
                "}"
        ],[
                "",
                "",
                ""
        ]
]
```

図14．変更後の実行結果

このように，変換で得られたプログラムの実行により試験を行うことで，仕様の不備な点の発見が行え，仕様の理解を助けることができる．

この変換手法で得られたPrologプログラムを実行した場合，得られる解は元の仕様を満たしているが，元の仕様を満たすすべての解が得られるとは限らない．しかし，元の仕様が仕様記述者の意図したように記述されているかどうかを，本手法で得られたプログラムを実行させることによっていくつかの解を得て試験したり，入力と出力を与えて真となるかどうかを試験するという目的には十分であり，こうしたデータ試験の結果により仕様記述者が仕様を修正することが可能となり，また再利用への道が開ける．

6.おわりに

今回は，TELL/NSLにおける静的記述から，プロトタイプとして実行可能なPrologプログラムを自動的な変換により直接的に得て，仕様に対するデータ試験をおこなうプロトタイピング手法について述べた．

この手法を用いて，ページのフォーマッティング問題，騎士巡回問題の他，電報解析問題，八王妃問題などについても，変換によりTELL/NSLの記述からPrologプログラムを得てデータ試験をおこない，TELL/NSLで記述された仕様の不備な点を発見でき，仕様を改善することができた．

今回の節やリテラルの並べ換えの変換で得られるプログラムにはいくつかの候補が存在し，最終的には人間によるプログラムの検査やデバッグが必要である．また，この変換は，仕様を直接的に変換しているため，仕様が戦略を含んで記述されていない場合は，得られたプログラムは一般的に実行に時間がかかる．特に，図４の騎士巡回問題の記述のように全ての組み合わせを調べる問題では，それが顕著である．実行効率を改善するために，現在，プログラム変換の研究の成果との結合を行い，変換の各段階でプログラムの効率化をおこなう変換の研究を行っている．

今回は，depth-first で直列実行するPrologの実行制御を考慮し，述語の入出力モードと入出力依存グラフを利用してリテラルや節の並べ換えを行っているが，並列実行が可能なPrologをターゲットにする場合にもその後の変換を交換するだけで，ここで用いられた入出力を利用することが可能である．

また，今回は述べなかったが，TELL/NSLによる動的仕様記述から，その意味として得られる時間論理式を介し，動作をシュミレートするPrologプログラムへの変換[7] を定めてあり，今後は静的と動的の変換方法を統合する予定である．

謝辞

御指導御討論をいただきましたTELLプロジェクトに参加された諸氏に感謝致します．

参考文献

[1] Enomoto,H. and Yonezaki,N. and Saeki,S.:"Natural Language Based Software Development System TELL," Proc. of ECAI-84,Pisa 1984,pp.721-731.

[2] Calaybrook,B.G.:"A Specification Method for the Specifying Data and Procedual Abstructions,"IEEE Tran. S.E., Vol.SE-8, No.5, pp449-459(1982).

[3] 佐伯，米崎，榎本:"自然言語の語彙分割による形式的仕様記述,"情報処理学会論文誌,Vol.25,No.2, 1984.

[4] 榎本，米崎，佐伯：" TELL/NSLおける語句定義構造と意味規則,"情報処理学会第30回全国大会予稿集,pp.485-486, 1985.

[5] Clocksin,W. and Mellish,C.:"Programming in Prolog, "1981, Springer Verlag.

[6] Peterson,J.L.:"Petri Net Theory and The Modeling of Systems,"1981, Prentice-Hall.

[7] 米崎，佐伯，市川，蓬莱，土井：" TELL/NSLおける動的記述のPrologへの変換,"情報処理学会第30回全国大会予稿集, pp.495-496,1985.