ソフトウェア基礎論 28－9
(1989. 3. 10)

動作ルーチン方式のソフトウェア生成系への新しい応用

New Applications of Action Routines to Software Generators

菅淳子　松野浩樹　徳田雄洋
Junko Kan, Hiroki Matsuno and Takehiro Tokuda

東京工業大学工学部情報工学科
Department of Computer Science, Tokyo Institute of Technology

あらまし

本報告では，動作ルーチン方式のソフトウェア生成系への新しい応用のための３つの方法について述べる．第１は，ＡＴＮ記法と呼ぶ，正規文法と動作ルーチンの組合せにより，プロトコルプログラムを記述する方法である．第２は，バックトラック方式により，対話的に動作ルーチンを実行する方法である．第３は，自然言語インターフェースのための演算子モデルによる構造エディタの構成法である．これらの方法に基づき，２つのソフトウェア生成系を試作した．１つはＡＴＮ記法によるプロトコルプログラム生成系であり，もう１つは自然言語からコマンド手続きを生成する構造エディタである．

Abstract

This paper gives three methods for improving the applicability of software generators based on action routines. Action routines are fragments of programs associated with each syntax rule to specify the semantics of syntax rules. Action routines are activated by external events such as reductions in the bottom-up syntax analysis.

The first method is the use of ATN (Augmented Transition Network) notation to generate protocol programs. The second method is the use of interactive action routines in terms of backtracking. The third method is the use of the operator model in constructing structure editors for natural language interface. We also present two examples of implementations of a protocol program generator and a command program generator using these methods.

1．はじめに

”ほとんどのソフトウェア生成は，
属性文法方式によるコーネル大学の
シンセサイザ生成系のコロラリにな
ってしまうのであろうか？”

1．1　意味定義法

構文主導型のソフトウェア生成における，代表的な意味定義法に，属性文法方式と動作ルーチン方式がある．属性文法方式では，属性の値の依存性を構文規則ごとに記述する．動作ルーチン方式では，局所的変数や大域的変数への代入動作等を行うプログラム片を構文規則ごとに記述する．

そして属性文法方式では属性の値の計算を，属性の値の依存性のみに基づいて行う．これに対し，動作ルーチン方式ではプログラム片の起動を，着目する外部イベントの発生順序に従って行う．

1．2　ソフトウェア生成

現在のところ，これらの方式によるソフトウェア生成系には，字句解析系生成系，構文解析系生成系，構造エディタ生成系，プロトコルプログラム生成系等がある．

これらの生成系により，生成しうるソフトウェアの範囲も，字句解析系，構文解析系，構造エディタ，プロトコルプログラムといったものから，ソースコード管理システム，ソフトウェアデータベース等に及んでいる．

以下代表的な生成系のシステム名を記す．

1）字句解析系
ＬＥＸシステム．．．正規式と対応する動作ルーチンを記述する．
2）構文解析系
ＹＡＣＣシステム．．文脈自由文法と対応する動作ルーチンを記述する．
ＧＡＧシステム．．．属性文法を記述する．
3）構造エディタ
ＡＬＯＥ生成系．．．終端記号，非終端記号，クラス，動作ルーチンを記述する．
シンセサイザ生成系．抽象構文，入力構文，出力規則（アンパース規則），変形規則，属性文法を記述する．
4）プロトコルプログラム
ＲＴＡＧシステム．．属性文法を記述する．

1．3　従来の動作ルーチン

従来の動作ルーチン方式では，生成規則に付随しているプログラム片を，次のような外部イベントの定める順序にしたがってを起動していた．

1）字句解析系
最長一致照合等により，1つの字句を獲得した時，プログラム辺を起動する．
2）上昇型構文解析系
上昇型構文解析により，生成規則の右辺を左辺に還元した時，プログラム辺を起動する．
3）下降構文解析系
下降型構文解析により，生成規則の左辺を右辺に展開した時，プログラム辺を起動する．
4）構造エディタ
ユーザのカーソルが抽象木の節点を訪問し，コマンドを発行した時，プログラム辺を起動する．

2．新しい動作ルーチン方式

本報告で行う3つの方法の提案は，以下の通りである．

1）正規文法と動作ルーチン
ＡＴＮ記法と呼ぶプロトコルプログラムの記述法による生成を提案する．
2）対話的動作ルーチン
バックトラックを行う動作ルーチン方式による対話的評価法を提案する．
3）構造エディタによる自然言語インターフェース
演算子モデルによる自然言語インターフェースの構成法を提案する．

このうち，紙数の関係で，先の2つの方法について比較的詳しく以下説明する．

2．1　正規文法と動作ルーチン

ＡＴＮ記法と呼ぶ記法を導入する．ＡＴＮ記

法によるプロトコルプログラムの記述は，後に見るように属性文法に基づくＲＴＡＧ法より簡潔な記述となる．

ＡＴＮ記法はＡＴＮ法からの自然な発想による記法であるので，まず，もとなったＡＴＮ法を説明する．ＡＴＮ法は，有限状態推移網（ＴＮ）法や再帰的有限状態推移網（ＲＴＮ）法の拡張であり，状態ｉから状態ｊへの推移関係を，次のように定義する．

［ＡＴＮ法の推移］

状態ｉで，条件Ｃ１，Ｃ２，．．，Ｃｎが成立するなら，入力記号ａを消費し，動作Ａ１，Ａ２，．．，Ａｍを順に起こし，状態ｊへ推移する．

ここで，推移条件Ｃ１，Ｃ２，．．．，Ｃｎは，入力記号，先読み，プログラムの変数の値の関係等の条件である．

状態ｉから状態ｊへの推移が開始されると，動作ルーチンの起動が起こる．動作としては，他のＡＴＮの呼び出しやプログラムの変数計算，代入，出力等がある．動作が終了すると，状態ｊへ移る．

このＡＴＮ法をもとに，ＡＴＮ記法を次のように提案する．

［ＡＴＮ記法］

ＡＴＮ記法は，ＡＴＮにおける正規文法，推移条件，動作を次のように表す．ただしイベント１，イベント２は終端記号である．

```
非終端記号０　：　　｛推移条件１｝
　　　　　　　　　　イベント１　非終端記号１
　　　　　　　　　　｛動作１｝
　　　　　　　｜　　｛推移条件２｝
　　　　　　　　　　イベント２　非終端記号２
　　　　　　　　　　｛動作２｝
　　　　　　　；
```

ＡＴＮ記法による記述の例を以下２つ示す．

［記述１］

言語｛ａ↑ｎ・ｂ↑ｎ｜ｎ≧１｝を認識するＡＹＮの例の記述を図２．１に示す．

```
－－－－－－－－－－－－－－－－－－－－－－
（変数の初期値）
Ａ０．ｖａｒ＝０，Ｂ０．ｖａｒ＝０．
－－－－－－－－－－－－－－－－－－－－－－
Ａ０　：　ａ　Ａ１
　　　　　｛ｕｐ（Ａ０．ｖａｒ）｝；
Ａ１　：　ａ　Ａ１
　　　　　｛ｕｐ（Ａ０．ｖａｒ）｝；
Ａ１　：　ｂ　Ｂ０
　　　　　｛ｕｐ（Ｂ０．ｖａｒ）｝；
```

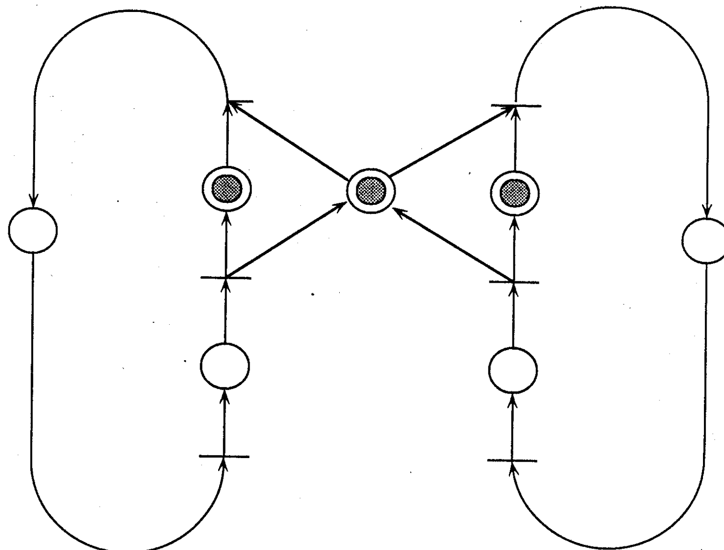

図２．２　ペトリネットの例

```
B0  :   b  B0
          {up(B0. var)};
B0  :   {A0. var=B0. var}
        e  C0
          {出力("yes")}
     |  {A0. var≠B0. var}
        e  C1
          {出力("no")};
---------------------
```

図2．1　言語認識の記述例

[記述2]

図2．2のペトリネットをATN記法で表した結果を図2．3に示す．7つの場所節点，6つの推移節点，16本の有向枝で，2つのプロセスの排他制御をモデル化したものである．

```
---------------------
(変数の初期値)
A0. var=1, B0. var=1,
C0. var=1, それ以外は0.
---------------------
A0  :   {A0. var≧1,
          C0. var≧1}
        e  A1
          {down(A0. var),
           down(C0. var),
           up(A1. var)};
A1  :   {A1. var≧1}
        f  A2
          {down(A1. var),
           up(A2. var)};
A2  :   {A2. var≧1}
        f  A0
          {down(A2. var),
           up(A0. var),
           up(C0. var)};
---------------------
B0  :   {B0. var≧1,
          C0. var≧1}
        e  B1
          {down(B0. var),
           down(C0. var),
           up(B1. var)};
B1  :   {B1. var≧1}
        g  B2
          {down(B1. var),
           up(B2. var)};
B2  :   {B2. var≧1}
        g  B0
          {down(B2. var),
           up(B0. var),
           up(C0. var)};
---------------------
```

図2．3　ペトリネットの記述

2．2　対話的動作ルーチン

バックトラックを行う動作ルーチン方式を提案する．この方式により対話的で効率的な動作ルーチンの実行が可能となる．ここでは上昇型の構文解析を仮定する．

[対話的動作ルーチン]

1．バックトラックを行う動作ルーチン方式を用いるために，入力域，出力域，スタック，変数の状態をすべてスタックに保存することにする．

2．入力列に変更が発生したら，はじめて右解析列の相違が生じるところの直前まで右解析列を戻す．入力域，出力域，スタック，変数の状態をもどし，そこから動作ルーチンの評価を再開する．

例として，算術式を表2．1の動作ルーチンで評価しよう．例えば算術式1＋2＊3に対応する右解析列は，64264631 0であり，この右解析列の規則番号が，得られて行く時に，対応する動作ルーチンを順に起動すると，表2．2のようになる．ただし記法"・"は，すぐ上のスタック要素と同一であることを示す．

表2．1　動作ルーチンの記述

```
---------------------
   文法規則              意味動作
---------------------
0) E' →  E
   Pop(op), 出力(op)
1) E  →  E+T
   Pop(op2), Pop(op1),
   Push(op1+op2)
```

2）E → T

動作なし

3）T → T＊F

Pop（op2），Pop（op1），

Push（op1＊op2）

4）T → F

動作なし

5）F → （ E ）

動作なし

6）F → n

Push（終端記号nが表す定数の値）

表2．2　動作ルーチンの実行

解析列	スタック
6	1
64	1 ・
642	1 ・ ・
・・・	
6426463 10	7 ・ 1 6 1 2 3 1 2 ・ 1 ・ ・

ここで入力列を変更してみる．例として，算術式を1＋2＋3に変更して評価しよう．この算術式に対応する右解析列は，6426416410である．64264までは共通であるので，ここまで状態を戻す．そして再実行する．この様子は表2．3のようになる．

表2．3　動作ルーチンの実行

解析列	スタック
64264	1 2 ・ 1 ・ ・
642641	3 1 2 ・ 1 ・ ・
・・・	
6426416410	6 ・ 3 3 ・ 3 1 2 ・ 1 ・ ・

一般の構造エディタの場合も，制限された動作ルーチンの使用が考えられる．

すなわち，動作ルーチンの起動を促す外部イベントを，構文解析系の与える順番のみとする．つまり後順たどりによる抽象木のたどりが開始され，その演算子の根の節点を通過した時，対応する動作ルーチンを起動する．

2．3　構造エディタによる自然言語インターフェース

構造エディタの考え方（抽象構文木からのアンパースにより具象構文が生ずる）は，演算子

```
event;          FrameArrival, CksumErr, TimeOut

begin
    NextFrameToSend := 0;
    FrameExpected := 0;
    FromHost(buffer);
    s.info := buffer;
    s.seq = NextFrameToSend;
    s.ack := 1 - FrameExpected;
    sendf(s);
    StartTimer(s.seq);
    repeat
        wait(event);
        if event = FrameArrival then
            begin
                getf(r);
                if r.seq = FrameExpected then
                    begin
                        ToHost(r.info);
                        inc(FrameExpected);
                    end;
                if r.ack = NextFrameToSend then
                    begin
                        FromHost(buffer);
                        inc(NextFrameToSend);
                    end;
            end;
        s.info := buffer;
        s.seq = NextFrameToSend;
        s.ack := 1 - FrameExpected;
        sendf(s);
        StartTimer(s.seq);
    untildoomsday
end;
```

図３．１　Ｐａｓｃａｌ風記述

```
<init> :        [U->DT] [N<-DT] <loop> .
        $0.NextFrame = 0
        $0.FrameExp = 0
        $0.Addr = $1.addr
        $0.Port = $1.port
        $2.addr = $1.addr
        $2.port = $1.port
        $2.seq = $0.NextFrame
        $2.ack = 1 - $0.FrameExp
        $2.info = $1.info
;
<loop> :        [N->DT] <pass> <ack> <process> <loop> .
        $2.seqmun = $1.seq
        $2.acknum = $1.ack
        <init>.Buffer = $1.info
        |       /timer/ <process> <loop> .
        $1.interval = TIMEOUT
;
<pass> :        [U<-DT] .
        if $0.seqmun == <init>.FrameExp
        $1.info = <init>.Buffer
        <init>.FrameExp = 1 - <init>.FrameExp
        |       .
;
<ack>  :        .
        if $0.acknum == <init>.NextFrame
        <init>.Buffer = #getdata#()
        <init>.NextFrame = 1 - <init>.NextFrame
        |       .
;
<process>       [N<-DT].
        $1.addr = <init>.Addr
        $1.prot = <init>.Prot
        $1.info = <init>.Buffer
        $1.seq = <init>.NextFrame
        $1.ack = 1 - <init>.FrameExp
;
```

図３．２　ＲＴＡＧ法の記述

```
<client> : <loop>
        {       next = 0
                expect = 0
                [U->FR]
                #strcpy ( Buff, FR.info );
        }
<loop> : <process>
        {       [N<-FR]
                        $ seq = next
                        $ ack = 1 - expect
                        $ ck_bit = 1
                timer [N]
                        $ interval = 10
                        $ number = next
        }
<process>:      FR_AR [N] <pass>
        {       [N->FR]
        }
                | T_OUT [N] <ack>
                { }
                | CK_ERR [N] <ack>
                { }
<pass> : <ack>
        {       if ( [FR].seq == expect ) then
                        [U<-FR]
                        expect = 1 - expect
                ;
                if ( [FR].ack == next ) then
                        [U->FR]
                        #strcpy ( Buff, FR.info);
                        next = 1 - next
                ;
        }
<ack> : <loop>
        {       #strcpy ( FR.info, Buff );
        }
```

図３．３　ＡＴＮ法の記述

```
%NTOP_data
SENTENCE        = [sentence]
                | "@[@0@n@]"
OUTPUT          = object device
                | "@1 wo@n@2 he@nsyutsuryoku suru."
SORT            = [file] key order
                | "@[@0, @]no@n@2 de@n@3 ni@nnarabekaeru."
SORT_RESULT     = [file] key order
                | "@[@0 @]no@n@2 de@n@3 ni@nnarabekaeta kekka"
FILE_OBJ        = [file] obj
                | "@[@0, @]no@n@2"
                        :
%TOP_data
FILE_NAME       = {alnum}
                | "@v"
NUMBER          = {digit}
                | "@v"
STRING          = {alnum}
                | "@v"
N_FIELD         = {digit}
                | "@v banme no field"
LINE_NUMBER     = {st}
                | "gyou bangou"
                        :
%CLASS_data
sentences       = SENTENCE .
sentence        = OUTPUT SORT .
object          = FILE_NAME FILE_OBJ SORT_RESULT .
device          = PRINTER FILE_NAME STD_OUT .
file            = FILE_NAME .
                        :
```

図3. 4　構文の記述

図3. 5　意味の記述

```
%{
char    s[512], s1[512], s2[512], s3[512], s4[512];
int     i, val;
}%
SENTENCE :
        pop(s1);
        for (i = 2; i <= Nfchild; i++) {
                pop(s2);
                sprintf(s,"%s;\n%s", s2, s1);
                strcpy(s1, s);
        }
        push(s1);
;;
OUTPUT :
        pop(s1);
        pop(s2);
        sprintf(s, "%s %s", s2, s1);
        push(s);
;;
SORT :
        pop(s1);
        pop(s2);
        pop(s3);
        for (i = 2; i <= Nfchild; i++) {
                pop(s4);
                sprintf(s, "%s %s", s4, s3);
                strcpy(s3, s);
                :
```

```
The Japanese Structure Editor for Interface    command_name :
file1, file2, no
1 banme no field, 2 banme no field, 1 banme no field to 2 banme no field no wa,
  wo
file3 he
syutsuryoku suru.
file3, no
< key > de
< order > ni
narabekaeru.
< sentence >

MODE:operator CLASS:key        11 operators     Selected Operator No. 1
 1. N_FIELD        2. ADD          3. DIFFERENCE      4. PRODUCT
 5. QUOTIENT       6. MOD          7. STR_LEN         8. EXP
 9. LOG           10. SQRT        11. INT
Input or Message
```

図3. 6　生成構造エディタの実行の様子

モデルという考え方をもたらす．

［演算子モデル］

自然言語の抽象構文は，抽象的な演算子からなる式とみなすことができる．具象構文を得るには，抽象木をアンパースすればよい．アンパースの仕方で，単純演算子と複合演算子に分けることができる．また単純演算子は更に，前置演算子，中置演算子，後置演算子，一般演算子に分けることができる．

演算子モデルは結果的に，自然言語処理における格文法の考え方と似てくる．また次のような特徴が発生する．

1）構造エディタの演算子選択による入力であれば，構文解析の問題が消滅する．

2）人工日本語を設定した場合，覚え難にくく書きにくいが，読みやすいという性質を利用できる，

3．　試作システムの例

3．1　プロトコルプログラム生成系

ここでは試作したネットワークシミュレータの例を示す．図3．1はTanenbaumの教科書にあるデータリンク層のプロトコル4のPascal言語風の記述である．これに対し図3．2はRTAGシステムによるプロトコル4の記述である．一方，図3．3はATN記法によるプロトコル4の記述である．

両者を比較すると，簡潔んことがよくわかる．

3．3．インターフェース用構造エディタの例

試作した日本語インターフェース用構造エディタの例を示す．ここでは，日本語を用いて，UNIXのコマンド手続きを作る構造エディタである．

図3．4は構文の記述で，図3．5は動作ルーチンによる意味の記述である．図3．6は生成された構造エディタの実行の様子である．コマンド手続きの詳細を知らなくとも，成績集計等のコマンド手続きを得ることができる．

4．おわりに

まず，ATN記法と呼ぶプロトコルプログラムの記述法に基づくプログラムの生成法の原理を示した．次に，バックトラックを行う対話型動作ルーチン方式の原理と，演算子モデルに基づく自然言語インターフェース用構造エディタの構成法の概略を示した．

属性文法方式から動作ルーチン方式への変換技術を利用すると，他にも効率的な動作ルーチンを得ることができる．

参考文献

[1] Anderson,D.P. and Landweber,L.H.: Protocol Specification by REAL TIME Attribute Grammers,Technical Reports, University of Wisconsin-Madison,1984.

[2]萩原正也，徳田雄洋：入力インターフェース用日本語サブセットの構成法とその応用，情報処理学会プログラム言語研究会，87-PL-13,1987.

[3] Staudt,B.J. et al.:The Gandalf System Reference Manual, Carnegie-Mellon University,1986.

[4] Tanenbaum,A.S: Computer Networks, Prentice-Hall, 1981.

[5] Teitelbaum,T. and Reps,T.:The Synthesizer Generator Refernce Manual,Cornell University,1985.

盛光印刷所