ICOT Technical Report: TR-367

TR-367

構文解析システムSAXのデバッグ環境

山崎重一郎、杉村領一
赤坂宏二、松本裕治

April, 1988

©1988, ICOT

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

構文解析システムＳＡＸのデバッグ環境

１．はじめに

ＩＣＯＴでは，談話理解システムの研究のベースとなるソフトウェアツール群を汎用日本語処理系（ＬＴＢ）として研究開発を進めている．ＳＡＸ［１］は，その中に含まれるツールの一つで，ＤＣＧ［２］の文法記述を対象とし，並列論理型言語による上昇型の並列構文解析を目的とする構文解析システムＡＸの逐次論理型言語版システムである．ＳＡＸは並列解析の擬似システムとしてのみでなく，逐次論理型言語による効率の良い全解探索構文解析システムとしても意味を持つ［３］．しかしながら，ＳＡＸによる解析は，一つの解析木を構成するための処理が，各段階で分散して処理されるために実行過程のトレースによるデバッグは極めて困難になる．このため，ＳＡＸの文法デバック環境は，解析実行による解析の履歴を，利用者の要求にしたがって取り出し，文法規則の適用過程として表示する環境として実現した．本論文では，まずＳＡＸによる解析の実行を概説し，次にＳＡＸのデバッグ環境の実現方法ついて述べる．

２．ＳＡＸによる解析の実行

ＳＡＸによる解析は，ボトムアップにある要素が構成されると，それを基にしてより大きな解析木を構成することによって進行する．このときの文法規則の選択は，規則の右辺の左隅要素によって決められる．そして新しい要素がボトムアップに構成されるたびに，選択された規則の２番目以降の要素が埋め合わせられて，完全な木が構成されていく．ＳＡＸでは，文法に現れる全ての単語および統語範疇が論理型言語の述語として表現されており，解析の進行による要素の構成は，述語の呼び出しに対応している．呼び出された述語には，新しい規則の選択と，すでに予想されている規則の充足の検査の２種類の役割がある．文法に規則の右辺の左隅要素として現れる要素は新しい規則の選択の役割を担うので，ＴＹＰＥ１要素と呼び，規則の右辺の２番目以降の要素は，規則の充足の検査の役割を担うので，ＴＹＰＥ２要素と呼ぶ．文法の複数の規則に現れる要素の中には，ＴＹＰＥ１でもＴＹＰＥ２でもあるような要素も存在する．また，規則の右辺の右隅の要素が充足されることは，予想されていた規則による解析木が完成したことを意味するので，その規則の左辺要素の呼び出しが行われる．したがって，規則の右辺右隅の要素は規則の左辺要素の呼び出しの役割を担うことになる．ＴＹＰＥ１で右隅要素でもある要素をＴＹＰＥ１０と呼び，ＴＹＰＥ２で右隅の要素をＴＹＰＥ２—ＬＡＳＴと呼ぶ．またＴＹＰＥ２で右隅要素でない要素をＴＹＰＥ２—ＭＩＤＤＬＥと呼び，右隅要素でないＴＹＰＥ１要素を狭義のＴＹＰＥ１と呼ぶ．ＳＡＸでは，文法の各規則の右辺にそれぞれの位置を表す識別子を付与し，各述語は，この識別子を使用して規則の選択や充足の検査を行う．

《例１》　次の文法には，識別子が付与されている．　id_1 ～id_5 が識別子である．

```
規則通番
    1：  n    ---->  [ｃａｒｓ].
    2：  ａｕｘ---->  [ｃａｎ].
    3：  ｖ    ---->  [ｂｅ].
    4：  ａ    ---->  [ｕｓｅｆｕｌ].
    5：  ｓ    ---->  ｎｐ, id_1  ａｕｘ, id_2  ｖｐ.
    6：  ｎｐ  ---->  ｎ.
    7：  ｖｐ  ---->  ｖ, id_3  ａ.
    8：  ｓ    ---->  ｎｐ, id_4  ｖｐ.
    9：  ｐｐ  ---->  ｐｒｅｐ, id_5  ｎｐ.
```

ＴＹＰＥ１

上の文法のｖは狭義のＴＹＰＥ１要素であり，次のような述語になる．

```
v(X, [id_3(X) | Y ], Y).
```

上の文法のｎはＴＹＰＥ１０であり，次のような述語になる．

```
n(X,Y1,Y2) :- np(X,Y1,Y2).
```

ＴＹＰＥ２

上の文法のａｕｘはＴＹＰＥ２—ＭＩＤＤＬＥであり，次のような述語になる．

```
aux( [] ,Y,Y ).
aux( [id_1(X) | Z ] , [id_2(X) | Y1] ,Y2) :- aux(Z,Y1,Y2).
aux( [_ | Z ] ,Y1,Y2) :- aux(Z,Y1,Y2).
```

ＴＹＰＥ１かつＴＹＰＥ２

上の文法のｎｐは狭義のＴＹＰＥ１かつＴＹＰＥ２－ＬＡＳＴであり，次のような述語になる．

```
np(X,Y1,Y3) :- np1(X,Y1,Y2),np2(X,Y2,Y3).
np1(X, [id__1(X),id__4(X)|Y2],Y2).
np2( [],Y,Y ).
np2( [id__5(X)|Z ],Y2,Y4) :- pp(X,Y2,Y3),np2(Z,Y3,Y4).
np2( [__|Z ],Y2,Y3) :- np2(Z,Y2,Y3).
```

ＳＡＸによる解析の実行は，入力文を構成するそれぞれの単語に対応する述語を次のように共有変数で結合して一斉に起動することによって開始される．

《例２》　入力文　　ｃａｒｓ　ｃａｎ　ｂｅ　ｕｓｅｆｕｌ．

に対する解析の実行は次のようにして行う．

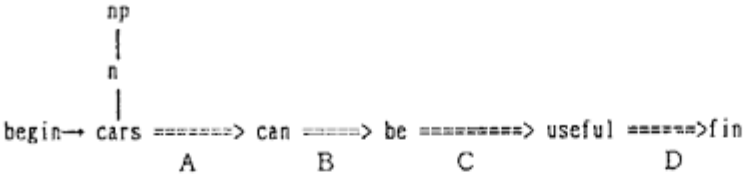
```
ｃａｒｓ（ｂｅｇｉｎ，Ａ，［］），
ｃａｎ（Ａ，Ｂ，［］），
ｂｅ（Ｂ，Ｃ，［］），
ｕｓｅｆｕｌ（Ｃ，Ｄ，［］），
ｆｉｎ（Ｄ）．
```

ここで，ｂｅｇｉｎは入力文の左端を表す識別子であり，ｆｉｎは解析結果を検査する述語である．

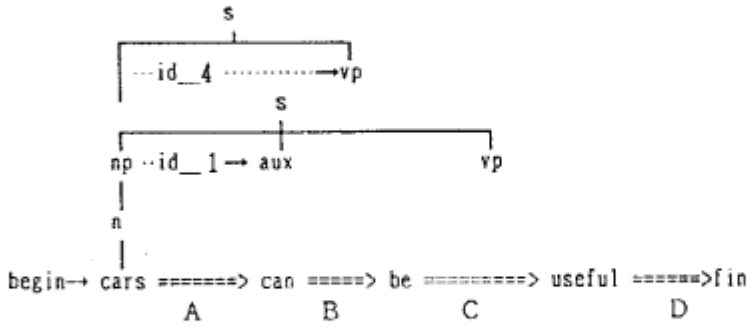
そして，解析の進行とともに，より上位の要素の述語が呼び出され，述語を繋いでいる共有変数は，それらの述語が出力した識別子によって具体化されていく．述語が共有変数を通じて入出力する識別子は，その引数として識別子のリストを持つような再帰的構造を持っており，解析の進行によって，より上位の木構造構成されていくことは，識別子とリストのより深い構造が具体化されていくことに対応している．

《例３》　ＳＡＸの解析において一つの解析木が構成される過程と共有変数の具体化の過程を例２の入力文の解析によって例示する．

① 述語 cars によっ述語ｎが起動され，述語ｎによって述語npが起動される．



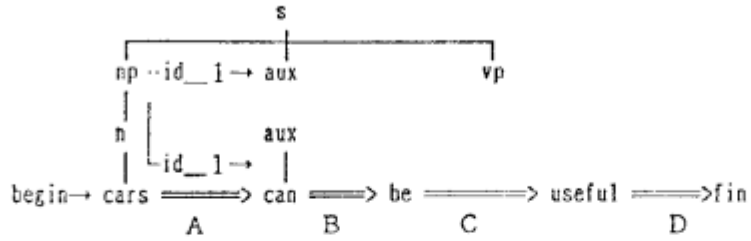
② ＴＹＰＥ１述語np　がid__１とid__4 を出力する．これは，次のような二つの木構造を予想することに相当する．



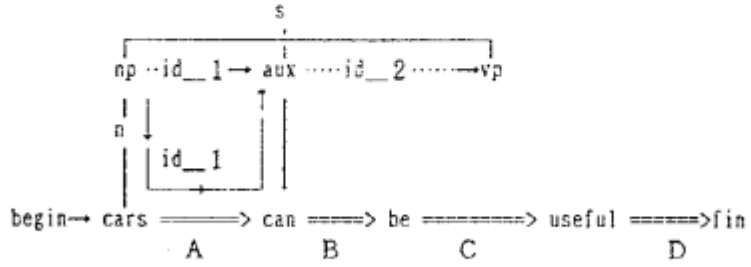
このとき共有変数Ａは　［id__１(begin),id__4(begin)］　に具体化される．

以下では，id__1 によって予想された木構造にのみ注目する．

③ 共有変数　Aが述語 cars の出力識別子によって具体化されることにより，述語 can　が述語 auxを起動する．

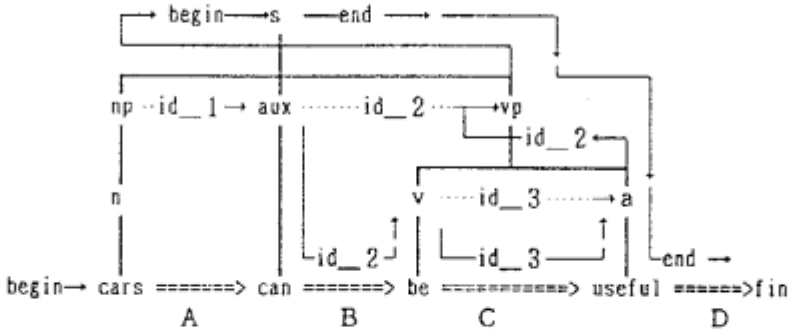


④ ＴＹＰＥ２—ＭＩＤＤＬＥ述語 auxは共有変数Ａを通じて渡ってきた識別子id_1をうけとると，　id_2を出力する．これは，　can　から構成された aux　が予想された文法規則の aux　に適合したことを意味しているので，　canから構成された auxの部分木が，その上の不完全な木の auxに直接されたことになる．



このとき共有変数Ｂは　［id_2 (begin) ］　に具体化される．

⑤ 同様にして　be，によってｖが起動されると，述語ｖは id _3 を出力する．このときbe　の入力の id _2 は，id_3 の引数として運ばれる．そしてuseful　が　ａ　を起動して，ａが id _3 の入力によりｖｐを起動するときに，ｖｐの入力として，運んでいた id _2 を渡す．述語ｖｐは　id_2を受け取ると，述語ｓを起動する．このときも同様に，id_2 の引数として運んでいた識別子ｂｅｇｉｎをｓ　の入力として渡す．述語ｓ　は，解析の目標となる要素なので，特別に予めｂｅｇｉｎを受け取るとｅｎｄを出力するＴＹＰＥ２—ＭＩＤＤＬＥ要素として定義されている．ｂｅｇｉｎはｓを構成する要素の左端を表す識別子であり，ｅｎｄは右端を表す識別子になっている．ｓの出力のｅｎｄはｖｐ，ａ，ｕｓｅｆｕｌの出力に順に具体化され，ｆｉｎとの共有変数Ｄに具体化される．このために，ｃａｒｓの入力にｂｅｇｉｎが具体化されていて，ｕｓｅｆｕｌの出力変数にｅｎｄが具体化されたことになるが，この状況は，ｃａｒｓからｕｓｅｆｕｌまでの間がｓを構成することを表している．



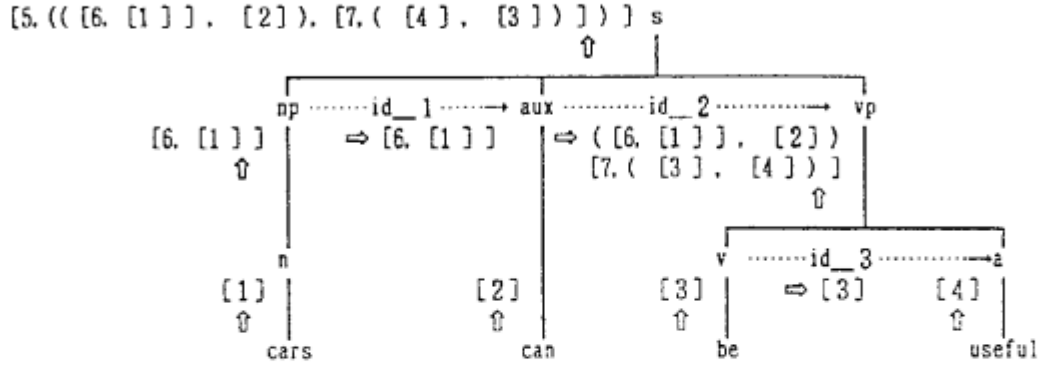
このとき共有変数Ｃは［id_3 (　［id_2 (begin) ］) ］に具体化され，共有変数Ｄは［ｅｎｄ］に具体化される．

3．解析結果の累積方法

解析の成功は，ｆｉｎが構成目標要素の右端を表す識別子（上の例での　ｅｎｄ）を検出することによって知ることができるが，実際の解析結果についての情報は，これだけではわからない．解析結果を知るためには，識別子の引数に変数を追加し，識別子によって文法規則の適用履歴を累積することが必要である．識別子による文法規則適用履歴の累積は，ＴＹＰＥ１０及びＴＹＰＥ２　ＬＡＳＴで成功した規則の規則通番を追加し，ＴＹＰＥ２　ＭＩＤＤＬＥ及びＴＹＰＥ２　ＬＡＳＴで識別子によって運ばれてきた規則適用履歴と配下からきた規則

適用履歴をマージし，狭義のＴＹＰＥ１では配下からきた規則適用履歴を出力識別子にのせて送ることによって実現できる．

《例４》　識別子による規則通番の累積例（数字は例１の文法の規則通番である）．



ここで，［C，（A，B）］は，規則Ａによる結果と規則Ｂによる結果が規則Ｃによってまとめられたことを意味する．ただし，規則Ｃが３項以上の規則によって構成される場合は，［C，((.,((A,X),Y),.),B)］となる．また，終端記号から構成されるときは［C］と記する．

このように，識別子を用いて解析履歴を累積すると，結果として文末の識別子（ｅｎｄ）にその解析の過程で使用された全ての文法規則の履歴が残されていることになる．

４．文法デバッグ環境

文法デバッグ環境は，識別子とＤＣＧ規則の位置の対応づけを行う機能を持ち，解析終了後に識別子の集まりとして共有変数に残されている解析の履歴を利用者の要求にしたがって取り出しそれをＤＣＧ規則の適応過程として表示する環境を与える．

４．１　識別子とＤＣＧ規則の対応づけ

ＳＡＸの実行結果として得られるのは識別子のリストであるが，利用者にとって必要なのはこれらの識別子が対応している文法規則の位置であり，識別子自体は必要ではない．このためにデバック環境では識別子に対応する文法規則の位置を求める機能を持ち，利用者が識別子を意識せずにデバックできるようにする．これは，識別子名を規則通番と要素位置の対によって構成し，この情報を利用することによってＤＣＧ規則の中の対応する位置を知ることで実現できる．

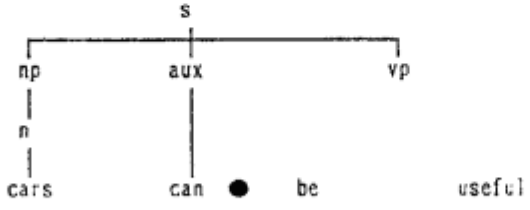
《例５》　識別子名の規約とＤＣＧ規則の対応

識別子名　　：　ｉｄ＿１２＿２　　（ｉｄ＿規則通番＿要素位置）
１２番目の規則　：　ａ　－－＞　ｂ，ｃ，ｄ，ｆ．
この場合，識別子名の情報により（識別子の位置を●で表示すると）

ａ　－－＞　ｂ，ｃ，●ｄ，ｆ．が得られる．

４．２　解析過程の表示方法

文法デバッグ環境は，基本的に，構文木の構成過程を表示する環境として実現する．しかしながら，実際の解析過程で現れる木構造は莫大な数になり，適切な選択手段を持たなければ実用にはならない．この選択手段の一つとして，入力文の指定した単語の区切りの位置より左側に構成される不完全な木構造を全て表示することを考える．これは，例３で　be　と　useful　の間を区切り位置とすると，次のような不完全な木を表示することに対応している．

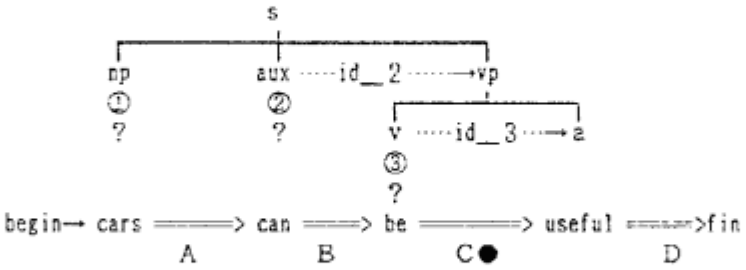


この木構造は cars によって予想された規則に can までが適合していることを示している．この木と同時に表示される他の木構造も can 迄が成功している木構造のみなので，この基準は表示される木構造の選択手段としてはたらく．

4．3　区切り位置の左側の不完全な木の表示方法

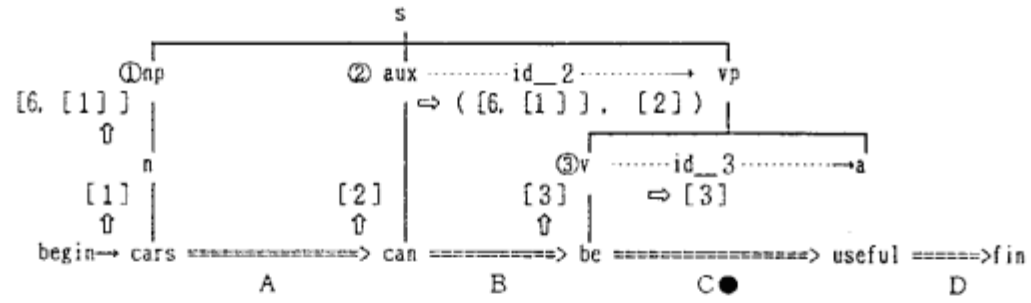
区切り位置の共有変数をモニタして得られる情報を，例3の解析木が構成されたときの共有変数Cをモニタして得られる情報によって説明する．

Cの中ではid__3が運ばれ，id__3の中でid__2が運ばれるのでCをモニタすることによって次のような木構造を得ることができる．



しかしながら，識別子のみの情報からでは，①や②や③に支配される部分木の情報を得ることはできない．これらの情報を得るために，識別子による文法規則の規則通番の累積を利用する．これによって，id__3で③配下の規則適用履歴がわかり，id__2によって①配下と②配下の規則適用履歴がわかるので，結果として，区切りの左側で構成される不完全な部分木が得られる．

C＝［id__3(［id__2(begin,(［6,［1］］,［2］))］,［3］)］



4．4　指定範囲から構成される完全な部分木の表示方法

上の不完全な木構造は useful によって a が構成されることを期待していることも示している．もし，このｓの解析が失敗していた場合，利用者は，解析の失敗の原因が a を構成する文法にあることを知ることができる．一般的に，区切り位置の左側に利用者が期待している解析木の構造が現れているにもかかわらず，解析に失敗している場合，その右側の要素を構成する文法に誤りが含まれていることがわかるので，誤り発生位置の特定は，区切り位置を少しずつ右に変更しながら調べていくことによって発見できる．しかし，この方法は，いかにも作業効率が悪そうである．このかわりに，利用者が指定した範囲で構成されている完全な部分木を全て表示できれば，その部分で利用者が期待している統語範疇が完成しているか否かを知ることができるので，誤り発生位置の範囲をもっと簡単に絞りこめる．このために，ＳＡＸのデバッグ環境に，指定範囲で構成された完全な部分木を得る機能をもたせることにした．指定範囲で構成された完全な部分木を得る方法は，基本的に解析目標の統語範疇まで完成した解析木を表示する方法と同じであり，指定範囲の左からｂｅｇｉｎに対応する識別子を入力して，指定範囲の右側の要素の出力側の共有変数で，その識別子と対をなすｅｎｄに対応する識別子を検出することによって実現できる．このために，すべての非終端記号に対して，次のようなＴＹＰＥ２　ＭＩＤＤＬＥの定義を追加する．

要素名（［left(N,LC)｜Z)］,［right(N,LC)｜Y1］,Y2,LCC) :- 要素名 (Z,Y1,Y2,LCC).

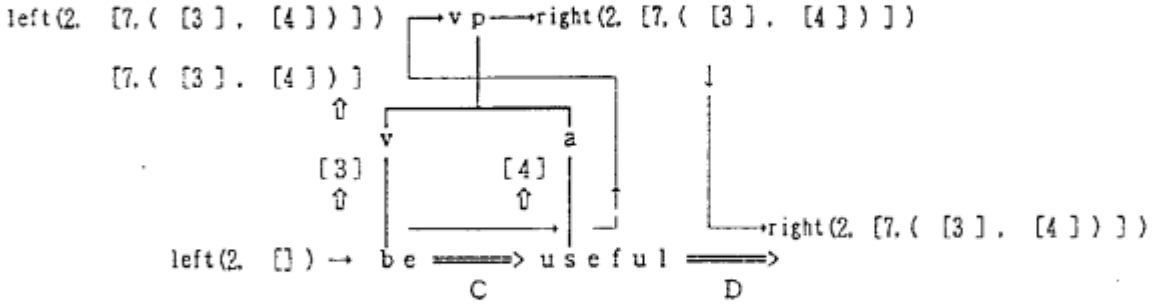
ただし，N は，入力文の単語の位置を表す番号であり，LCおよびLCC は規則の適用履歴を累積するための変数である．

そして，解析の起動を次のように変更する．

《例6》　入力文　cars　can　be　useful　に対して

```
cars(begin( [] ),A, [] ),
can(    [left(1,  [] ) |A ],B,  [] ),        %  単語の位置  1
be(     [left(2,  [] ) |B ],C,  [] ),        %  単語の位置  2
useful( [left(3,  [] ) |C ],D,  [] ),        %  単語の位置  3
fin(D).
```

このとき，ｂｅ　と　ｕｓｅｆｕｌ　の範囲から構成される完全な部分木の情報を得たいとする．



この場合，共有変数Ｄの中に，right(2, [7,([3], [4])]) を発見することによって　ｂｅ　から　ｕｓｅｆｕｌの間で上のような完全な木構造が構成されていることがわかる．

5．おわりに

ＳＡＸのデバッグ環境の実現方法について述べたが，今後，さらに使用経験をもとに，より使い易い環境を実現したい．

【参考文献】

[1] 松本裕治，杉村領一：論理型言語に基づく構文解析システムＳＡＸ，ソフトウェア科学会論文誌，1986, Vol3, No4

[2] Pereira, F. C. N. and Warren, D. H. D. :Definite Clause Grammars for Language Analysis - A survey of the Formalism and a Comparison with Augmented Transition Networks, Artif. Intel., Vol. 13(1980)

[3] Okunishi, Sugimura, Matsumoto: Comparison of Logic Programing Based Natural Language Parsing System