ショートノート

対話形式による構文誤りの修正†

宮本衛市†† 北山泰英††† 梶川 登†††

プログラムの構文上の誤りといえども，それを形式的に修正する場合には一般に任意性があり，プログラマの是認を必要とすることから，対話形式によりプログラマの確認を得ながら誤りを修正していく PASCAL パーザを開発した．このパーザは誤りの検出・修正のためのいくつかの方式を有しており，まず修正案を提示してその是非を問う．いくつかの修正案では修正しきれないときに初めて編集モードに入り，プログラマ自身による字句編集あるいは構文編集を行う．このパーザはコンパイラを始め，プログラミング・システムにおけるソーステキストの読込みなどに適用することができる．

1. はじめに

プログラムの構文上の誤りを自動的に修正する試みがいろいろと提案されているが(たとえば文献1),2)),構文上の誤りといえどもその修正には一般に任意性があり，プログラマが結局は最終的な判断をしなければならない．バッチ処理のもとでは計算機側で一方的な処置をせざるをえないが，最近の TSS をはじめとするソフトウェア環境では，プログラマは計算機と相対して処理ができるようになり，誤りを検出すると対話的に修正を求める構文解析システムも開発されている[3]．

プログラムの構文上の誤りを形式的に修正する場合には，一般にいくつかの案が考えられる．そこで，われわれは妥当と思われる順にいくつかの修正案を提示してプログラマに是否を問い，それで修正しきれないときに初めてプログラマ自身による編集に修正を委ね，再び解析を続行する PASCAL パーザを開発した．これはコンパイラにおける構文解析を始め，PASCAL に立脚したエディタ[4]，デバッガ[5]などで，誤りを含むソーステキストの読込みに適用することができる．

2. 誤りの検出と修正案の作成

パーザは再帰的下向き構文解析を基本とし，パーザ自身では後戻りの解析を行わないので，誤りをできるだけ早く検出するために，以下に述べるいくつかの方式を導入している．

2.1 構文解析からの誤りの検出と修正

パーザは後述するように字句が名前の場合を除き，読み込んだ字句をもとにして解析の針路を決定し，書換え規則で決められた字句集合に含まれない字句を読み込んだときに，構文上の誤りとして検出する．誤り回復法[6]によれば，ここで適当なメッセージを出力し，読捨て中止字句集合に含まれる字句が現れるまでテキストを読み捨て，再起を図るのであるが，本方式ではテキストを先読みして字句を収集し，適用すべき書換え規則を推定する．

一般に，非終端記号αの書換え規則を次式に示すように，選択的な規則 $\beta_i(1\leq i\leq n(\alpha))$ の集まりと考える．

$$\alpha ::= \beta_1 | \beta_2 | \cdots | \beta_{n(\alpha)} \qquad (1)$$

そこで各非終端記号に対し，誤り回復法と同様に先読み中止字句集合 $S_{\mathrm{stop}}(\alpha)$ を設定し，さらに各書換え規則 $\beta_i(1\leq i\leq n(\alpha))$ ごとに必要字句集合 $S_{\mathrm{nec}}(\alpha,i)$ および十分字句集合 $S_{\mathrm{suf}}(\alpha,i)$ を設定する．ただし，$S_{\mathrm{stop}}(\alpha)$ に含まれる字句が現れるまでテキストを先読みしたとき，もしその字句列が規則 β_i から生成されているのならば，$S_{\mathrm{nec}}(\alpha,i)$ は規則 β_i にとって必須な字句の集合であり，$S_{\mathrm{suf}}(\alpha,i)$ は規則 β_i から導出される字句列に含まれる字句の集合である．そこで，非終端記号αの書換えで誤りを検出したとき，$S_{\mathrm{stop}}(\alpha)$ に含まれる字句が現れるまでテキストを先読みして先読み字句集合 S_{in} を求め，次のようなペナルティを各規則ごとに求める．

$$P(i)=\mathrm{card}(S_{\mathrm{nec}}(\alpha,i)-S_{\mathrm{in}}) + w\cdot\mathrm{card}(S_{\mathrm{in}}-S_{\mathrm{suf}}(\alpha,i)) \quad (1\leq i\leq n(\alpha)) \qquad (2)$$

ここで，card は集合の要素の数を求める関数であり，

† Interactive Correction of Syntax Errors by EIICHI MIYAMOTO (Division of Information Engineering, Graduate School of Engineering, Hokkaido University), YASUHIDE KITAYAMA and NOBORU KAJIKAWA (Mitsubishi Electric Corp.).

†† 北海道大学大学院工学研究科情報工学専攻

††† 三菱電機(株)

表 1 〈文〉の分類を対象にした字句集合
Table 1 Token sets for the classification of 〈statement〉.

文の種類	S_{nec}	S_{suf}
代入文	[becomes]	expsys+S_{nec}+stateclosys
手続き呼出し文	[ ]	expsys+S_{nec}+stateclosys
if 文	[ifsy, thensy]	expsys+S_{nec}
case 文	[casesy, ofsy]	expsys+S_{nec}
while 文	[whilesy, dosy]	expsys+S_{nec}
for 文	[forsy, becomes, tosy, dosy]	expsys+S_{nec}
for 文	[forsy, becomes, downtosy, dosy]	expsys+S_{nec}
with 文	[withsy, dosy]	expsys+S_{nec}
goto 文	[gotosy, numsy]	S_{nec}+stateclosys

S_{stop}=blockbegsys+statebegsys
+[thensy, elsesy, ofsy, dosy, untilsy,endsy, semicolon]
expsys=[numsy, chrsy, strsy, idsy, lparen, lbracket, rparen, rbracket, notsy, relop, addop, mulop, arrow, period, comma, dperiod, nilsy]
stateclosys=[endsy, semicolon, untilsy, elsesy]
blockbegsys=[labelsy, constsy, typesy, varsy, procsy, functsy, beginsy]
statebegsys=[beginsy, ifsy, casesy, whilesy, repeatsy, forsy, withsy, gotosy]

右辺第1項は規則 β_i を適用したとき脱落している字句に対するペナルティを，第2項は余分な字句に対するペナルティを意味し，w は前者に対する後者の重み係数である．

そこで，あるペナルティ以下のペナルティをもつ規則を適用可能な規則と仮定し，その規則を適用するために必要な字句の修正案を提示する．すなわち，誤りを引き起した字句を t_e，t_e に続く字句を t_f，規則 β_i の先頭にくる字句を $t_r(\alpha,i)$ とすると，次のような修正案を提示してプログラマの確認を求める．

a) $t_e \notin S_{suf}(\alpha,i)$，かつ $t_f=t_r(\alpha,i)$ のとき
'delete 〈t_e〉?'

b) $t_e \notin S_{suf}(\alpha,i)$，かつ $t_f \neq t_r(\alpha,i)$ のとき
'replace 〈t_e〉 with 〈$t_r(\alpha,i)$〉?'

c) $t_e \in S_{suf}(\alpha,i)$，かつ $t_f=t_r(\alpha,i)$ のとき
'interchange 〈t_e〉 with 〈t_f〉?'

d) $t_e \in S_{suf}(\alpha,i)$，かつ $t_f \neq t_r(\alpha,i)$ のとき
'insert 〈$t_r(\alpha,i)$〉?'

ただし，字句はソースイメージに直して出力する．

上記の方式を誤りの検出後の処理ばかりでなく，文の選択における代入文のように，字句が名前であることで選択される規則を含む非終端記号の構文解析にも適用する．なぜなら，構文解析上の名前は定義・宣言される各種の名前のいわば総称であって，綴りのまちがった予約語も字句解析上名前に分類してしまうので，予約語を含む特殊記号で分類することにくらべ，名前であることで選択した規則が誤りにつながる危険性が大きいからである．

いま，〈文〉の分類を例にとると，まず**表1**に示すような字句集合を設定する．ただし，複合文または **repeat** 文はそれぞれ **begin** または **repeat** の脱落に対し，対応する **end** または **until** が現れるまで誤りが発見されず，本方式では検出できないので除いてある．

そこで，次の文

$a>0$ **then** $a:=a+1$;

のような **if** の脱落に対しては，先読み字句集合

S_{in}=[idsy, relop, numsy, thensy]

が得られ，各文のペナルティは

p(if)=1, p(assign)=2, p(while)=3, …(ω=1とする)

となり，まず 'insert if?' を提示する．他の予約語の脱落に対しても同様にして推定できるが，**while** または **with** の脱落に対しては **while** 文と **with** 文に対するペナルティは同じになり，指示される修正案に基づいてプログラマが選択することになる．なお，予約語の綴りのまちがいに対しては，次章で述べるように構文解析に入る前に検出して修正案を提示する．

本方式は書換え規則から導出される特徴的な字句の有無に基づいており，それらが脱落または改変しては当然適確な判断はできなくなる．先読みした区間に2ないし3個以上の誤りがあれば，どの規則にとってもペナルティが大きくなり，結局修正案の提示はしない．また，字句を単位として解析するため，後述の図1の例にあるように，字句の分離の失敗につながるような誤りに対しては効果がない．さらに，本方式自身では構文解析の後戻りはしないので，たまたま誤った書換え規則を選択したときには，その先で解析に窮してしまうことになる．そのようなときは後述の編集モードに入り，プログラマによる編集によって，パーザを正常な箇所にまで復帰させる．

2.2 意味処理からの誤りの検出

PASCAL では変数の書き方や演算記号はデータ型に応じて決められている．そこで，〈変数〉や〈式〉を解析するときには，字句に応じて解析した後に意味解析をするのではなく，むしろ意味解析から書換え規則を選択してデータ型に応じた構文解析を行うことにより，意味解析に基づく誤りの検出と修正の処理が容易になる．

2.3 テキストの段付けからの誤りの検出

PASCAL には **begin** に対する **end**, **repeat** に対する **until**，左かっこに対する右かっこなどのように

構文を囲む記法が多く用いられ，しかも，閉じる側では対応する相手を明示しないので解析上は内側から閉じていくことになり，閉じる字句を書き忘れるとプログラムの構造は著しく乱されてしまう．通常，プログラムは段付けされて書かれており，段付けはプログラムの入れ子構造に対し示唆的な情報をになっている．段付けには何ら規則はないが，内側の入れ子は段付けして書き出すものと仮定し，段付けの情報を活用する．たとえば複合文の場合，複合文を構成する各文が複合文を抱える文より段付けされているものとすれば，複合文の解析中，列挙された文の書き出しが複合文を抱える文より右側になければ，**end** の書き忘れの可能性がある．もちろんこれには文法的な決め手はないが，'insert end?' と警告を発し，プログラマの注意を喚起することができる．

2.4 字句の綴りの誤りの検出

字句解析単独では特殊文字からなる特殊記号の綴りの誤りは検出できるが，予約語および名前の綴りの誤りは一般的には検出できない．テキスト上で英数字の綴りからなる字句が現れたときには，予約語，名前の定義および名前の参照の3通りが考えられる．そこで，予約語をもつ書換え規則があるのに名前が読み込まれたとき，あるいは名前の参照があったのに定義されていなかったときには，それぞれ該当する可能性のある予約語表または名前表をもとにして，読み込んだ字句の綴りと，たとえば1文字の誤入力を仮定すれば一致するような予約語または名前を探索する[7]．

3. プログラミング・システムへの組込み

われわれはこれまでPASCAL向けの編集[4]，虫取り[5]，およびそれらを総合した支援システム[8),9)]を開発してきたが，システムへ入力するプログラムやコマンドは誤りのないことが前提であった．そこで，誤りを含んだテキストでも読み込めるように，前章で述べた方式を組み込んだ対話形式のパーザを作成した．このパーザがもつコマンドを**表2**に示すが，パーザには構文解析モードと字句編集モードがある．

構文解析モードでは前章で述べた方式を活用するが，基本的には 2.2〜2.4 節の三つの方式がそれらの得意とするところでまず誤りの検出を行い，それでも残った誤りに対し 2.1 節の方式を適用する．すなわち，〈変数〉と〈式〉の解析は 2.2 節で述べた意味解析に基づいて構文解析を進める．2.3節の方式は複合文，**repeat** 文，仮パラメータ部などの繰返しを含む構文に適用する．また，字句が名前のときには選択しうる書換え規則がもつ予約語を対象に該当しそうな予約語を探す．次に，字句が名前以外であれば再帰的下向き構文解析に従い，誤りが

表2 コマンドの一覧

Table 2 Summary of commands.

モード	コマンド	機能
構文解析	空行	他の修正案の提示要求
	Y	修正案を了承
	G	誤りではないことの指示
	E	字句編集に入ることの指示
字句編集	B [〈token〉]	指定した字句まで戻る
	I {〈token〉}	字句の挿入
	R {〈token〉}	字句の置換
	D	字句の削減
	F [〈token〉]	指定した字句まで進める
	Q	字句編集の終了

〈token〉: 文字列，[ ]: 省略可，{ }: 1回以上の繰返し

```
   1PROGRAM PASCAL(OUTPUT);
   2 CONS C=100;
 *** ^CORRECT 'CONS' TO 'CONST' ?
>Y
   3 TYPE PTR=@NODE;
   4      NODE=RECORD KEY:INTEGER; NEXT:PTR END;
   5 VAR S N:INTEGER; P:PTR;
 ***        ^INSERT ',' ?
>Y
   6 PROCEDURE FACT(MAX:INTEGER;
 ***                            ^INSERT ')' ?
>Y
   7  VAR O:PTR;
   8 BEGIN N:=0; P:=NIL ; S:=1;
 ***         ^DELETE ':' ?
>Y
   9  WHILE MAX>S THEN
 ***               ^REPLACE 'THEN' WITH 'DO' ?
>Y
  10   BEGIN N=N+1 S:=S8N;
 ***           ^REPLACE '=' WITH ':=' ?
>Y
 ***                ^INSERT ';' ?
>Y
 ***                    ^CORRECT 'S8N' TO 'SIN' ?
>E
 >>>EDITING NODE, NOW 'S8N'
>R S * N
>0
      BEGIN N:=N+1; S:=S*N;
  11   NEW(O); O@KEY:=S; O.NEXT:=P; P:=O;
 ***               ^INSERT '.' ?
>Y
 ***                          ^INSERT '@' ?
>Y
 ***                                        ^INSERT 'END' ?
>Y
  12  S>1 THEN S:=S DIV N
 ***  ^INSERT 'IF' ?
>Y
  13 END;
  14BEGIN FACT(O) WRITELN(N,S) END.
 ***               ^INSERT ';' ?
>Y
```

図1 対話による誤り修正の例

Fig. 1 An example of interactive error correction.

検出されれば2.1節の方式を適用する．一方，字句が名前であればただちに2.1節の方式を適用するが，名前で始まる規則のペナルティが0であれば解析を続行する．

構文解析モードで，修正できない場合には字句編集モードに入り，プログラマの指示に基づき，誤りが検出された字句の近傍を，字句を単位として修正する．修正後は，修正した字句を含めて構文解析を再開できる手続きにまで戻って解析を続行する[3]．なお，名前の定義の修正等はシステムのもつ編集機能を活用する[8),9)]．

図1はプログラムの修正をタイプライタ型端末を通して行った例であり，これまで述べてきた方式を駆使している．とくに，行番号 10 では二つの誤りが重なったため，';' までの先読みで正常な代入文と見なし，'=' で誤りを検出した．この場合は ':=' を期待するだけであるが，一応ペナルティを次の字句集合に基づいて計算した上での修正案の提示である．

S_{stop}=factbegsys+[plus, minus]

S_{nec}=[becomes]

$S_{suf}=S_{nec}+S_{stop}$

ここで，factbegsys は〈因子〉を開始させる字句の集合である．

4. あとがき

対話形式を前提とすれば，誤りの検出のために大胆な方式の導入も可能であり，修正案は必ずしも一つである必要はない．2.1 節の方式は誤りを検出した後の事後処理としてばかりでなく，字句の先読みに基づく大局的判断で行う，誤りを含んだテキストの構文解析と見ることもできる．その場合の能力等の理論的解析は今後の課題である．

謝辞 北海道大学工学部竹村伸一教授からは本研究を進めるに当たって貴重なご教示をいただき，また桃内佳雄助手からは本文をまとめるに当たり懇切なご助言をいただいた．ここに記して深謝申し上げます．

参考文献

1) Fischer, C. N., Milton, D. R. and Quiring, S. B.: An Efficient Insertion—Only Error—Correction for LL(1) Parsers, Fourth ACM Symposium of Principles of Programming Languages, pp. 97-103 (1977).
2) Tai, K. C.: Syntactic Error Correction in Programming Languages, *IEEE Trans. Softw. Eng.*, Vol. SE-4, No. 5, pp. 414-425 (1978).
3) 菅井賢三，岡根幸宏，荒木俊郎，都倉信樹：対話型修正編集機能を持つ PASCAL 構文解析システム，情報処理学会論文誌，Vol. 22, No. 2, pp. 148-154 (1981).
4) 宮本衛市，浅見可津志：プログラム構造に基づいた編集機能をもつテキスト・エディタ，情報処理学会論文誌，Vol. 20, No. 6, pp. 474-480 (1979).
5) 宮本衛市，堀川博司，高橋幸伸：虫取り支援を目的とした PASCAL プログラム実行制御システム，情報処理学会論文誌, Vol. 22, No. 5, pp. 424-433 (1981).
6) Ammann, U.: Error Recovery in Recursive Descent Parsers, Berichte des Instituts für Informatik, ETH Zürich, Nr. 25 (1978).
7) Muth F. E., Jr. and Tharp, A. L.: Correcting Human Error in Alphanumeric Terminal Input, *Information Proccessing & Management*, Vol. 13, No. 6, pp. 329-337 (1977).
8) 宮本衛市，中川礼一：PASCAL 実行編集制御システム，情報処理学会第 23 回全国大会，pp. 305, 306 (1981).
9) 宮本衛市，竹村伸一：PASCAL プログラミングのための支援環境システム，北海道大学工学部研究報告，第 108 号，pp. 81-92 (1982).

(昭和 57 年 7 月 9 日受付)
(昭和 57 年 11 月 8 日採録)