Tratamento de Erros Sintáticos em A.S.D. com implementação de Procedimentos Recursivos

Error Report, *Recovery and Repair*

Erros Sintáticos

- Tipos:
  - ausência de um símbolo: var x: integer
  - Símbolo mal escrito: bigin -> será reconhecido como id
  - Excesso de símbolos: while x > y then do

- Felizmente, a maioria dos erros são simples
  - Pesquisa com estudantes de Pascal
    - 80% dos enunciados contém apenas um erro; 13% tem dois
    - 90% são erros em um único token
    - 60% são erros de pontuação: p.ex., uso do ponto e vírgula (;)
    - 20% são erros de operadores e operandos: p.ex., omissão de : no símbolo :=
    - 15% são erros de palavras-chave: p. ex., erros ortográficos (wrteln)

Erros Sintáticos

- Em muitos compiladores, ao encontrar uma construção mal formada o erro é reportado e a tarefa da Análise Sintática é dada como concluída

- Mas na prática o compilador pode e até deve reportar o erro e tentar continuar a Análise Sintática
  - para detectar outros erros, se houver, e assim diminuir a necessidade de recomeçar a compilação a cada relato de erro.

- A realização efetiva do tratamento de erros pode ser uma tarefa difícil
  - O tratamento inadequado de erros pode introduzir uma avalanche de erros espúrios, que não foram cometidos pelo programador, mas pelo tratamento de erros realizado

Três processos, geralmente, são realizados nesse ponto:

1) Notificação

2) Recuperação (modo de pânico):

   pula-se parte subseqüente da entrada até encontrar um token de sincronização (porto seguro para continuar a análise)

3) Reparo (recuperação local):

   faz-se algumas suposições sobre a natureza do erro e a intenção do escritor do programa.

   Altera-se 1 símbolo apenas: despreza o símbolo, ou substitui ele por outro ou ainda insere um novo token.

   A opção mais comum é inserir 1 símbolo (comum para ; faltantes)

1) a localização de um erro sintático é notificada

2) se possível, os tokens que seriam uma continuação válida do programa são impressos

3) os tokens que podem servir para continuar a análise são computados. Uma seqüência mínima de tokens é pulada até que um destes tokens de continuação seja encontrado

4) a localização da recuperação (ponto de recomeço) é notificada

5) a análise pode ser chaveada para o “modo reparo” também.

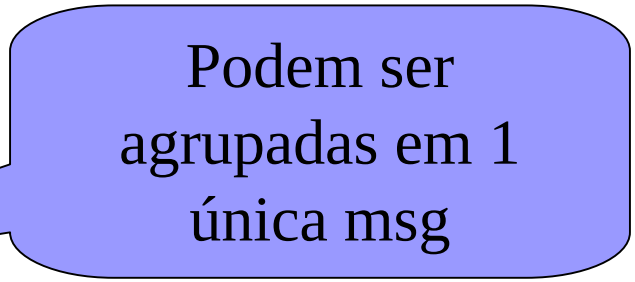
   Neste ponto, o analisador se comporta como usual, exceto que nenhum token de entrada é lido.

   Ao invés, uma seqüência mínima (geralmente um símbolo) é sintetizada para reparar o erro.

   Os tokens sintetizados são notificados como símbolos inseridos. Depois de sair do modo reparo, a A.S. continua como usual.

Exemplo em Modula-2:

```
Module test;
   Begin
                    |16
  ..IF..(a=.].1.write.(a) end;
                |12|14
   End test.
```



Error messages

3, 12: error syntax error
3, 12: expected symbols: ident, integer, real, string, char, ...
3, 14: restart point
3, 16: error syntax error
3, 16: restart point
3, 16: repair inserted symbol ‘)’
3, 18: repair inserted symbol ‘then’

Existem regras e suposições para o tratamento de erros sintáticos.

Elas são selecionadas de acordo com:

1. a concepção da linguagem e
2. o método de construção do A Sintático

Regra da palavra-chave

Antes de tudo, é claro que o reconhecimento é facilitado ou somente possível quando a estrutura da linguagem for simples.

Em particular, se depois de diagnosticar um erro, alguma parte subseqüente da entrada deve ser pulada (ignorada), então é necessário que a linguagem contenha palavras-chaves

- para as quais seja improvável o uso incorreto e essas possam servir para trazer o compilador a um ponto seguro.

Regra “não haverá pânico”

É característico da A.S.D. com procedimentos recursivos que a análise seja dividida em procedimentos que podem chamar-se mutuamente para que a análise se complete.

Se um procedimento analisador detectar um erro, ele não deveria meramente se recusar a continuar e dizer o que aconteceu ao seu analisador principal.

Ao invés disso, deveria ele próprio continuar a olhar o texto até o ponto onde a análise possa ser retomada.

Regra “não haverá pânico”

A conseqüência dessa regra é que não haverá saída de um procedimento a não ser desse ponto regular de término.

Uma possível interpretação dessa regra consiste em:

1. pular o texto de entrada ao detectar uma formação ilegal até o próximo símbolo que pode seguir corretamente o procedimento correto.
2. Isso implica que cada analisador deverá conhecer seu conjunto de símbolos seguidores no lugar de sua ativação.

Refinamentos da A.S. para a Recuperação de Erros:

- cada procedimento deve conhecer o conjunto de seus seguidores no local de ativação
- cada procedimento tem um parâmetro fsys que especifica seus seguidores (follows)
- no fim de cada procedimento, um teste é incluído para verificar se o próximo símbolo está no conjunto de follows
- vamos aumentar esse conjunto com os first de uma construção para que o procedimento teste não consuma muito da entrada e tenha um desempenho melhor nos casos em que o programador esqueceu somente um símbolo
- os seguidores são inicializados com as palavras chaves do procedimento e vão sendo gradualmente acrescidos para símbolos seguidores legais quando penetramos na hierarquia dos procedimentos de análise

```
procedure teste(S1, S2, n)
S1=conjunto de próximos símbolos; se o símbolo correto
      não está entre eles, há um erro
S2=conjunto de símbolos adicionais de parada cuja
      presença é um erro, mas que não devem ser ignorados
n=diagnóstico de erro

begin
      if not(símbolo in S1) then
         begin
         erro(n);
         S1:=S1+S2;
         while not(símbolo in S1) do
                  símbolo:=analex(S);
         end
end
```

teste também pode ser chamado no começo de um procedimento de análise para verificar se o símbolo corrente é um símbolo inicial admissível.

Os casos são:

$$A::=a_1S_1 \mid a_2S_2 \mid ... \mid a_nS_n \mid X$$

Podemos usar dentro de X (no começo) e S1 deve ser igual ao First(X) e S2 é o Follow(A)

➔ no caso de nada seguirX

Reparo

Essa estratégia de uso de teste é infeliz no caso em que o erro é por omissão de um símbolo. Vejam o que também pode ser feito no caso do <comando_composto>

<comando_composto>::=
begin <comando> {; <comando>} end

```
if símbolo = sbegin then
begin
        símbolo:=analex(S);
        comando([';' , end]+fsys);
        while símbolo in [';']+First(comando) do
        begin
                if símbolo=';' then símbolo:=analex(S)
                else reparo("; inserido");
                comando ([';' , end]+fsys);
        end;
        if símbolo=send then símbolo:=analex(S)
        else erro("end esperado");
end
```

Procedimento de impressão de erro

Conclusão

- Deve ficar claro que nenhum esquema que com razoável eficiência traduz seqüências corretas
  - deveria TAMBÉM ser hábil a manusear todas as possíveis construções incorretas.

- As características de um bom compilador são:
  - Nenhuma sequência de entrada deveria colocar o compilador em colapso
  - Todas as construções ilegais são detectadas e reportadas
  - Erros que ocorrem com freqüência são diagnosticados corretamente e não causam muitas mensagem de falsos erros.

Linguagem MICRO: exercícios

1) Levante o conjunto dos seguidores de cada não terminal da gramática acima
2) Modifique o Analisador Sintático fornecido para que ele trate erros sintáticos.

    Siga a estratégia dada em aula que consome símbolos estranhos ao contexto para retornar a um ponto seguro de análise e apresente também alguns reparos.

Resposta do Exercício 1 sobre MICRO

Gramática da linguagem MICRO em notação EBNF

<programa> ::= <bloco> .
<bloco> ::= <decl> inicio <comandos> fim
<decl> ::= [tipo <idtipo>] [var <idvar>]
<idtipo> ::= <id> = <id> ; {<idtipo>}
<idvar> ::= <id> : <id> ; {<idvar>}
<comandos> ::= <coms> { ; <coms>}
<coms> ::= <id> := <expr> |
read (<listaid>) |
write (<listaexp>) |
if <exp> then <coms> [else <coms>] |
inicio <comandos> fim
<exp> ::= [+|-] <termo> { (+|-) <termo>}
<termo> ::= <fator> {(*|/) <fator>}
<fator> ::= <id> | <numero> | (<exp>)
<listaid> ::= <id> {, <id>}
<listaexp> ::= <exp> {, <exp>}
OBS: <id> e <numero> são considerados terminais.

Follow

<programa> = {$}
<bloco> = {.}
<decl> = {inicio}
<idtipo> = {var, inicio}
<idvar> = {inicio}
<comandos> = {fim}
<coms> ={; , fim, else}
<exp> = {;, fim,), , , then, else}
<termo> = {+, -, ; , fim,), , , then, else}
<fator> = {*, /, +, -, ; , fim,), , , then, else}
<listaid> = {) }
<listaexp> = {) }

Resposta do Exercício 2

- Entra direto em:
  - Bloco, comandos e fator: usar TESTE no começo destes
  - Usar reparo em comando composto

AS com procedimentos recursivos

```
Begin {programa principal}
    simbolo := analex(s);
    programa([eof]);
    If símbolo <> eof then erro(12)
End.
Procedure programa(fsys: set of codigo);
Begin
    bloco([ponto]+fsys);
    Se simbolo = ponto then simbolo:= analex(s) else erro(11);
    Teste(fsys,[],'eof esperado')
End;
```

```
Procedure bloco (fsys: set of codigo); {engloba todos os procedimentos}
...
begin {bloco}
  teste([stipo, svar, sinicio],[ FIRST COMANDO],'declaração de tipo, var ou início esperados');
   decl([sinicio] + fsys);
   if símbolo = sinicio then símbolo := analex(s) else erro(10);
   comandos([sfim] + fsys);
   if símbolo = sfim then símbolo := analex(s) else erro(9);
  Teste(fsys,[],'ponto ou eof esperados')
  end;
```

```
Procedure comandos (fsys: set of codigo);
Begin
    teste([FIRST COMS],[ ], 'Início, Read, Write, If ou
    Identificador esperados');
    coms([pontovirgula, sfim]+fsys);
    while símbolo in [pontovirgula]+First(coms) do
    begin
        if símbolo= pontovirgula then símbolo:=analex(s)
        else reparo("; inserido");
        coms ([pontovirgula, sfim]+fsys);
    end
End;
```

```
Procedure coms(fsys: set of codigo);
Begin
    If simbolo = sread then …
    Else
    if simbolo = ident then  …
    Else
    if símbolo = swrite then…
    else
    f símbolo = sif then …
    else
    if simbolo = sinicio then …
    Teste(fsys, [FIRSTCOMS], 'comando, ; ou fim esperados');
End;
```

Eliminamos a mensagem de erro com este TESTE:

erro('comando esperado')

```
Procedure listaid(fsys: set of codigo);

    Begin
        Repeat
            Simbolo := analex(s);
            If símbolo = ident then simbolo := analex(s) else erro(1);
            Teste([virgula,fechapar],fsys, ‘, ou ) esperados’)
        Until simbolo <> virgula
    End;
```

Como EXP tem teste, não precisa chamar aqui

```
Procedure listaexp(fsys: set of codigo);

    Begin
        Repeat
            Simbolo := analex(s); Exp([virgula, fechapar]+ fsys)
        Until simbolo <> virgula
    End;
```

```
Procedure exp(fsys: set of codigo);
  Procedure termo(fsys: set of codigo);
    Procedure fator(fsys: set of codigo);  {observem o relaxamento da regra FATOR}
      Begin
            Teste(FIRSTFATOR, fsys, 'identificador, ( ou número esperados);
            while simbolo in FirstFator do
            begin

                  Case simbolo of
            Ident, numero: simbolo := analex(s);
                  abrepar: begin
                        simbolo := analex(s); exp([fechapar]+fsys);
                        if símbolo = fechapar then símbolo := analex(s) else erro(6)
                     end
                  end;
                  Teste(fsys,FIRSTFATOR,'identificador, ( ou número esperados);
            end;
      begin { termo}
            fator([multi,divi] + fsys);
             while símbolo in [mult, divi] do
                  begin
                        simbolo := analex(s); fator([multi,divi] + fsys)
                  end
      end;
            begin {exp}
            if símbolo in [mais, menos] then símbolo := analex(s);
            termo([mais,menos]+fsys);
            while símbolo in [mais, menos] do
                  begin
                        símbolo := analex(s); Termo([mais,menos]+fsys)
                  end
       end;
```

```
Procedure decl(fsys: set of codigo);
{observem o relaxamento da regra DECL}
    Begin
        repeat
        If simbolo = stipo then
        Begin
                Simbolo := analex(s); Idtipo([svar,sinicio]+fsys);
        End;
        If símbolo = svar then
        Begin
                Simbolo := analex(s); Idvar([sinicio]+fsys);
        End;
        teste(fsys, [FIRSTDECL + FIRSTCOMANDO],'tipo      ou
        var esperados');
        until simbolo in [FIRSTCOMANDO]
    End;
```

```
Procedure idtipo (fsys: set of codigo);
    Begin
        Repeat
            If símbolo = ident then simbolo := analex(s) else erro(1);
            If simbolo = igual then simbolo := analex(s) else erro(2);
            If simbolo = ident then simbolo := analex(s) else erro(1);
            If simbolo = pontovirgula then simbolo := analex(s) else erro(3)
        Until simbolo <> ident;
        Teste([svar,sinicio],FIRSTCOMS,'var ou inicio esperados')
    End;

Procedure idvar (fsys: set of codigo);
    Begin
    Repeat
        If simbolo = ident then simbolo := analex(s) else erro(1);
        If simbolo = doispontos then simbolo := analex(s) else erro (4);
        If símbolo = ident then simbolo := analex(s) else erro(1);
        If simbolo := pontovirgula then simbolo := analex(s) else erro(3)
    Until simbolo <> ident;
   Teste([sinicio],FIRSTCOMS,'inicio esperado')

    End;
```