

2.8 - Recuperação de Erros

- ♦ Um compilador deve tentar encontrar todos os erros possíveis num programa fonte, para minimizar o número de vezes que o programa é submetido ao compilador até estar livre de erros. Estes erros são os que impedem a geração de código (podemos dizer que a tarefa do compilador estará completa ao gerar código). Os erros de execução, embora importantes, saem do escopo da compilação. O que o compilador pode fazer, nesse caso, é fornecer dados para que, ao acontecer um erro na execução, possa se achar o comando do programa fonte responsável por tal erro (o que não é uma tarefa fácil).
- ♦ Recuperação de erro é o processo de determinar como continuar a analisar um programa fonte após um erro ter sido encontrado.
- ♦ Os erros podem ser divididos, de acordo com a fase da compilação em que são detetados, em: erros léxicos, erros sintáticos e erros semânticos.
- ♦ Ao ser encontrado um erro, o compilador pode ter os seguintes tipos de reação:
 - a) **"Crash" ou "loop"**: Um programa errado, contendo uma combinação imprevista de circunstâncias, provoca no compilador um "crash" ou um "loop". Se entendermos um compilador como uma função definida em V_T^* (vocabulário terminal), devemos assegurar que essa função é total, ou seja, o compilador não poderá provocar um "crash" ou entrar em "loop" qualquer que seja a entrada.
 - b) **Produzir saída inválida** (aparentemente tudo estando certo): Esse comportamento é o mais indesejável de todos.

Recuperação de Erros

- c) **Desistir:** Embora seja uma reação honesta, não é muito desejável (o programa vai precisar ser "compilado" várias vezes até estar totalmente livre de erros).
 - d) **Recuperar e continuar a análise:** Ignorar o trecho errado do programa e continuar analisando como se nada tivesse acontecido (pode-se "desligar" a geração de código, uma vez que não será executado). É interessante ignorar o menor trecho possível para que o trecho ignorado não seja causa de outros erros. Isso é especialmente verdadeiro nas declarações, onde sua omissão causa várias mensagens do tipo "identificador não declarado".
 - e) **Corrigir:** É o que pode revelar o maior número de erros, além de ser possível (pelo menos, em princípio) gerar código para um programa errado. As alterações feitas no programa fonte devem ficar bem explícitas para que o programador não aceite uma "correção" errada.
- ♦ Nesse capítulo vamos nos concentrar na reação (d).

Erros léxicos

- ♦ Sendo Γ o conjunto de caracteres permissíveis e V_T , o vocabulário terminal podemos descrever a função do analisador léxico como a função:

$$S : \Gamma^* \rightarrow V_T$$

Recuperação de Erros

- ♦ Um erro léxico ocorre quando encontramos $\alpha \in \Gamma^*$ tal que $S(\alpha)$ não é definido. Isso pode ocorrer devido a:
 - a) $x \in \Gamma$ tal que x é um caractere inválido para S
 - b) identificador ou constante mal formados.
- ♦ Recuperações possíveis:
 - a) ignorar o caractere inválido (como se não existisse)
 - b) substituir o identificador ou constante mal formados por um identificador ou constante "fantasma".

Em termos do "scanner":

- relatar erro e posição aproximada no comando fonte
- devolver ao analisador sintático o par (SYN, SEM) onde SYN é <id> ou <cte> e SEM é uma palavra especial para indicar a característica de "fantasma".

Correção ortográfica

- ♦ Existem várias situações em que um compilador pode suspeitar que um identificador foi escrito incorretamente. Nestes casos o compilador deve comparar o identificador com os da tabela de símbolos e tentar decidir qual dentre estes foi escrito incorretamente.

Recuperação de Erros

- ♦ Frequentemente, devido a incorreções ortográficas, um identificador aparece somente uma ou duas vezes, com nenhuma atribuição de valor ou com nenhuma referência a seu valor. Isso pode ser verificado facilmente por meio de um contador de atribuições e um contador de referências para cada identificador da tabela de símbolos. As entradas com um dos contadores igual a zero seriam candidatas a correção ortográfica.
- ♦ Cerca de 80% dos erros ortográficos (evidência empírica) em programas pertencem a uma das categorias:
 - a) um caractere errado
 - b) um caractere faltando
 - c) um caractere extra inserido
 - d) dois caracteres adjacentes permutados

Portanto, muitos erros de ortografia podem ser corrigidos por verificação dessas quatro categorias de erro. Para limitar o número de entradas da tabela de símbolos contra as quais deve-se verificar o identificador incorreto, pode-se usar o contexto no qual esse identificador é usado (por exemplo: se o identificador incorreto está sendo usado como um rótulo, devem ser selecionadas apenas as entradas da tabela de símbolos relativas a rótulos). Um teste mais simples é no tamanho do identificador: como serão testados somente as quatro categorias acima, selecionar apenas as entradas de tamanho $n-1$, n e $n+1$, onde n é o tamanho do identificador incorreto.

Recuperação de Erros

Recuperação de erros sintáticos

- ♦ Em qualquer ponto da análise sintática, o programa fonte tem a forma xTt onde x é a parte já processada, T é o próximo símbolo e t , é o resto do programa. Nesse ponto, detectado um erro, devemos determinar como alterar o programa para "apagar" o erro. As seguintes tentativas poderão ser feitas:
 - a) remover T e tentar analisar novamente;
 - b) inserir uma cadeia de terminais q entre x e T (produzindo $xqTt$) e recomençar a análise a partir de qTt . Essa inserção deve permitir processar toda a cadeia qT antes que um novo erro ocorra;
 - c) inserir uma cadeia q qualquer entre x e T , e recomençar a análise a partir de Tt ;
 - d) remover alguns símbolos mais à direita de x .

As tentativas (c) e (d) são maneiras ruins de recuperação porque uma vez que x já foi processado, a informação semântica associada a x já foi tomada. Adicionar q a x ou retirar parte de x significa alterar a informação semântica associada, o que não é fácil de ser feito. Os métodos (a) e (b), portanto, devem ser os preferidos.

Recuperação de erros em analisadores "top-down"

- ♦ Vamos assumir um analisador sintático "top-down" que trabalha sem retorno (ou executa análises alternativas em paralelo ou usa algum contexto para decidir a regra correta a ser aplicada em cada passo).

Recuperação de Erros

- Em qualquer passo da análise, uma ou mais árvores sintáticas foram construídas, com alguns ramos incompletos.

- Exemplo:

Gramática

$P \rightarrow A;$

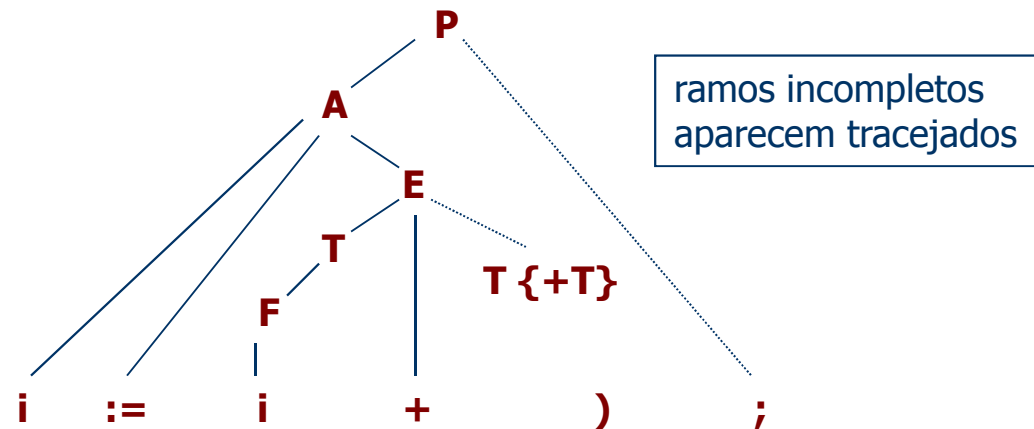
$A \rightarrow i := E$

$E \rightarrow T \{ + T \}$

$T \rightarrow F \{ * F \}$

$F \rightarrow i \mid (E)$

Árvore sintática para $i := i +)$;



- Um ramo incompleto U qualquer corresponde à aplicação de uma regra:

$$U \rightarrow x_1 x_2 \dots x_{i-1} x_i \dots x_n$$

onde $x_1 x_2 \dots x_{i-1}$ corresponde à parte completa do ramo e $x_i \dots x_n$ à parte incompleta. Por exemplo, na figura acima o ramo incompleto P corresponde à aplicação da regra $P \rightarrow A;$ onde A é a parte completa e $;$ a parte incompleta. O ramo incompleto E corresponde à aplicação da regra $E \rightarrow T \{ + T \}$. Para completar esse ramo precisamos de um T seguido por número qualquer de $+T$. A parte incompleta é, portanto, $T \{ + T \}$.

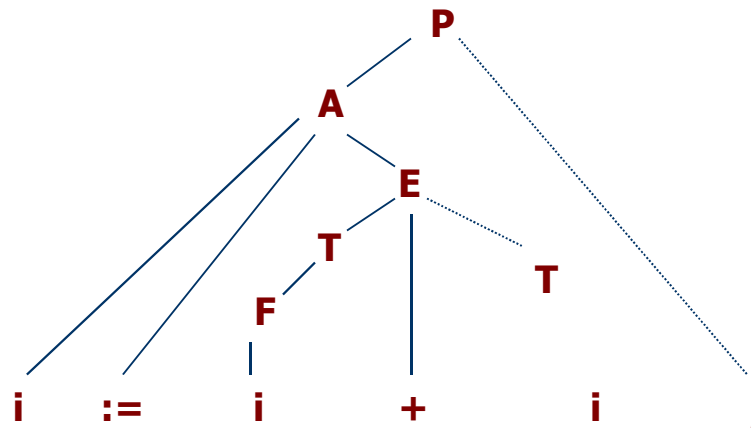
Recuperação de Erros

- ♦ As partes incompletas têm um grande papel na recuperação de erro pois elas mostram o que pode ou deve aparecer em seguida no programa fonte. Em termos gerais, para recuperação de erros devemos tomar as seguintes ações:
 - a) construir uma lista L de símbolos de partes incompletas;
 - b) examinar e descartar continuamente o símbolo T de Tt até encontrar um símbolo tal que $U \Rightarrow^* T...$ para algum $U \in L$;
 - c) determinar qual é o ramo incompleto responsável pela inclusão de U em L;
 - d) determinar uma cadeia terminal q de forma que inserida antes de T permita que a análise continue até ser completado o ramo determinado no passo (c);
 - e) inserir q e retornar a análise a partir do seu primeiro símbolo.
- ♦ No exemplo anterior, o erro foi detectado quando:

$$\underbrace{i := i + }_x \underbrace{)}_T \underbrace{;}_t \qquad L = \{ +, T, ; \}$$

O passo (b) do algoritmo irá descartar o símbolo $)$. O ramo que fez com que $;$ fosse incluído em L é $P \rightarrow A;$. Devemos completar o ramo $E \rightarrow T \{ + T \}$. A cadeia de terminais mais simples (notar que existem várias: $i, i+i, i+i+i, \dots$) é i . Após a recuperação teremos:

Recuperação de Erros



- ♦ e a análise é retomada com:

$$\underbrace{i \ := \ i}_{x} \ + \ \underbrace{i}_{T} \ ;$$
- ♦ Quando a implementação da análise "top down" é o recursivo descendente, as árvores sintáticas construídas não são representadas explicitamente, de forma que esse método não é inteiramente possível (notar que no passo (d), para determinar q, devem ser analisados todos os ramos incompletos que pertençam a subárvores de um dado ramo incompleto).
- ♦ Entretanto, em qualquer ponto da análise, cada procedimento que estiver sendo executado representa um ramo incompleto da árvore sintática. Assim, é possível escrever uma rotina de recuperação de erro para cada procedimento (a idéia é procurar um símbolo "especial" na sequência de entrada e ir para o ponto correspondente no procedimento envolvido).

Recuperação de Erros

- ♦ Exemplo: Seja um erro ocorrido dentro de uma lista de identificadores, cuja sintaxe é definida por:

$\langle \text{lista_id} \rangle ::= \langle \text{id} \rangle \{ , \langle \text{id} \rangle \} ;$

Nesse caso, procura-se por "," ou por ";". No primeiro caso, continua-se na rotina $\langle \text{lista_id} \rangle$. No segundo caso, deve-se sair de $\langle \text{lista_id} \rangle$.

Recuperação de erros em analisadores "bottom-up"

- ♦ Na recuperação "top down", muita informação sobre o que deve aparecer em seguida no programa fonte pode ser encontrada na árvore sintática parcialmente construída. Essa informação, no entanto, não existe tão prontamente disponível no caso de análise "bottom-up". Logo, como não podemos usar facilmente o contexto global (como todos os ramos incompletos da análise "top down"), devemos trabalhar apenas com o contexto local que envolve o ponto de erro.
- ♦ Vamos supor que o programa fonte está na forma xTt onde a parte já processada x é tal que $x = x_2x_1$ onde x_1 é o "head" do lado direito de uma regra (os símbolos de x_1 são os do topo da pilha). A recuperação deve ser feita somente por meio da retirada de T ou da inserção de uma cadeia terminal q entre x_1 e T . Isso pode ser feito da seguinte maneira:
 - a) se existir uma regra $U \rightarrow x_1zT...$ na gramática, deve ser inserida uma cadeia terminal q tal que $z \Rightarrow^* q$;

Recuperação de Erros

- b) se existir uma regra $U \rightarrow x_1 V \dots$ tal que $V \Rightarrow^+ z T \dots$ na gramática, deve ser inserida uma cadeia terminal q tal que $z \Rightarrow^* q$;
- c) se existir uma regra $U \rightarrow \dots V T \dots$ tal que $V \Rightarrow^+ \dots w z_2$ e $w \rightarrow x_1 z_1$ é uma regra da gramática, deve ser inserida uma cadeia terminal q tal que $z = z_1 z_2 \Rightarrow^* q$;
- d) se nenhum dos três casos anteriores não se aplica, retirar T .

♦ Exemplos:

caso	x_1	T	regras	z	q
1	()	$F \rightarrow (E)$	E	i
2	begin	:=	$\langle cs \rangle \rightarrow \text{begin } \langle sl \rangle \text{ end}$ $\langle sl \rangle \Rightarrow^* i := E$	i	i
3	$T *$)	$F \rightarrow (E)$ $E \Rightarrow^* T$ $T \rightarrow T * F$	F	i
4	array	step	retirar STEP		

- ♦ Esse método apresenta uma série de problemas:
 - a) se mais que uma regra se aplica, qual deve ser aplicada?
 - b) qual cadeia de terminais q tal que $z \Rightarrow^* q$ deve ser inserida?
 - c) é garantido que uma inserção não vai levar a uma sequência infinita de inserções?

Recuperação de Erros

- ♦ Um outro método (admitidamente primitivo) é o seguinte: manter um array STOPIT de símbolos especiais como: ":", "end", ".", ... Quando um erro é detetado as seguintes ações são tomadas:
 - a) os símbolos de Tt são examinados e descartados até que seja encontrado um símbolo de STOPIT;
 - b) após o passo (a) tem-se um novo símbolo T. Examinar e descartar símbolos do "tail" de x até que esse novo T possa seguir legalmente o que restar de x.
- ♦ Esse método viola a regra de não alterar x por causa das ações semânticas já tomadas. Seria melhor inserir uma cadeia de terminais que quando processada causasse reduções na pilha, o que, no entanto, é difícil de ser feito.

Recuperação de erros semânticos

- ♦ Vamos descrever um método simples de recuperação de erros semânticos (erros do programa fonte relacionados com o uso incorreto de identificadores e expressões). As idéias principais são as seguintes:
 - a) A recuperação consiste em trocar uma expressão ou identificador incorreto por um "correto". Isso é feito inserindo uma nova entrada na tabela de símbolos, com atributos determinados pelo contexto no qual ocorreu o erro.

Recuperação de Erros

- b) Mensagens de erro devidas a uma recuperação errada ou incompleta devem ser suprimidas onde for possível. Existem dois problemas principais: a supressão de mensagens extras devidas a um único erro e a supressão de mensagens duplicadas devido ao mesmo erro ocorrendo várias vezes.

Supressão de mensagens extras

- ♦ Em muitos casos onde um identificador é usado incorretamente, podemos mostrar uma mensagem de erro e continuar. Por exemplo, se um comando $A := B$ é analisado e A é real e B , booleano, podemos mostrar a mensagem " A e B não são de tipos compatíveis" e continuar a análise porque nenhuma ação semântica necessita estar associada ao não-terminal <atribuição> e portanto mensagens extras não serão impressas.
- ♦ Por outro lado, seja o caso de uma variável subscrita $A[i_1, \dots, i_n]$. Vamos supor que A não foi declarado como um array e que mostramos uma mensagem de erro e continuamos a analisar os subscritos. Ao final, o número de subscritos será verificado contra o número de dimensões de A , o que irá produzir uma nova mensagem de erro pois A não é um array.
- ♦ Para suprimir as mensagens extras, o identificador errado deve ser trocado por um "correto" com as seguintes precauções:

Recuperação de Erros

- uma nova entrada é inserida na tabela de símbolos com os atributos preenchidos tão corretamente quanto possível;
- o nome associado ao novo identificador deve ser um que não possa aparecer num programa fonte e ele deve ser reconhecível como um nome que foi inserido para corrigir um erro;
- a rotina de emissão de mensagens de erro deve verificar se o erro está relacionado a um identificador inserido "artificialmente" ou não. No primeiro caso, nenhuma mensagem é mostrada.

Supressão de mensagens duplicadas

- Mensagens de erro duplicadas ocorrem porque o mesmo identificador é usado incorretamente várias vezes. Isso pode ocorrer devido a uma declaração que não pôde ser analisada corretamente. Isso ocorre também numa linguagem com estrutura de bloco onde, devido a um erro, a estrutura do bloco ficar prejudicada.

```
(1)  begin real A; ...  
(2)    begin boolean A; ...  
(3)      begin ... end;  
(4)      A := A and B; ...  
(5)    end;  
(6)  end;
```

Devido ao erro em "begin" (linha 3) o "end" na linha 3 irá terminar o bloco da linha 2 e, portanto, na linha 4 será exibida uma mensagem como "A é real mas deveria ser booleano".

Recuperação de Erros

- Mensagens duplicadas podem ser suprimidas facilmente:
 - a) se um identificador não declarado é usado, inserir esse identificador na tabela de símbolos como descrito anteriormente.
 - b) associado à entrada de um identificador armazenar uma lista de maneiras incorretas de uso do identificador. Quando um identificador é usado incorretamente, percorrer essa lista: se o identificador já foi usado dessa maneira anteriormente, não exibir mensagem de erro.

Rotina de tratamento de erro semântico

- 1) se a mensagem N é "identificador não declarado" executar o passo 2. Caso contrário, ir para o passo 3.
- 2) inserir na tabela uma entrada ID com tipo T; armazenar o endereço em P; exibir a mensagem e retornar.
- 3) se P é o endereço de uma entrada inserida no passo 2 ou 7 então retornar (isso suprime mensagens extras).
- 4) se a mensagem N não se relaciona com o tipo errado da entrada P então exibir a mensagem e retornar.
- 5) se a entrada P já foi usada incorretamente da mesma maneira então ir para o passo 7.
- 6) exibir a mensagem e se $P \neq 0$, adicionar o uso incorreto à lista da entrada P.
- 7) se não existe entrada com identificador "corrigido" do tipo T, construir uma e inserir na tabela; atualizar P para essa nova entrada e retornar.

Parâmetros:

N - número da mensagem de erro

ID - identificador que causou o erro

P - endereço na tabela de símbolos que descreve o identificador incorreto

T - tipo que o identificador ou expressão deveria ter (com base no contexto no qual o erro ocorreu)