Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

Error Recovery in Predictive Parsing

- An error is detected during predictive parsing when
  1. The terminal on top of the stack does not match the next input symbol
  2. Nonterminal A is on top of the stack, a is the next input symbol, and $M[A, a]$ is error (i.e., the parsing-table entry is empty)
- **Panic Mode**
  - ***Panic-mode error recovery*** is based on the idea of skipping over symbols on the input until a token in a selected set of **synchronizing tokens** appears
  - Its effectiveness depends on the choice of synchronizing set

-
اگر کامپایلر به اون ارور رسید اون ارور رو به کاربر بده ولی متوقف نشه و ادامه بده و بقیه برنامه رو بره و بعد خارج بشه که کاربر بخواد برنامشو صحیح بکنه
توی پارس به خطا می خوره:
1- اگر حالتی پیش بیاد که توی استک ترمینال داشته باشیم و توی ورودی هم ترمینال داشته باشیم و عین هم نباشن این میشه ارور چون رسیدم به یک ترمینال که نمی تونیم حذفش کنیم
2- توی استک یک نان ترمینال داریم و توی ورودی ترمینال داشته باشیم ولی توی جدول پارس به ازای اون نان ترمینال و ترمینال حرکتی نداشته باشیم --> ارور داریم باز
Panic Mode یک روش رایج است که همه کامپایلرها استفاده می کنن --> این حالت بر این مبنا است که نادیده بگیریم یا حذف کنیم ینی وقتی به یک اروری خورد یا از ورودی بره جلو و به یک حالتی برسه که بتونه ادامه بده ینی یکسری کاراکتر اون وسط حذف کنه یا از استک یه چیزی رو حذف کنه که به یک حالت معتبر برسه و به ارور نخوره
توی این حالت synchronizing tokens تعریف میشه که میگه اگه ورودی رو داری اسکیپ میکنی تا جایی پیش برو و اسکیپ کن تا برسی به یک حالت معتبر --> پس حذف کن تا به یکی از این سیمبل هایی که توی مجموعه synchronizing tokens برسی

Error Recovery in Predictive Parsing

- **Panic Mode**
  1. Place all symbols in $FOLLOW(A)$ into the synchronizing set for nonterminal A
  2. If we add symbols in $FIRST(A)$ to the synchronizing set for nonterminal A, then it may be possible to resume parsing according to A if a symbol in $FIRST(A)$ appears in the input
  3. If a nonterminal can generate the empty string, then the production deriving ϵ can be used as a default
     - Doing so may postpone some error detection
  4. If a terminal on top of the stack cannot be matched, a simple idea is to pop the terminal

-
ممکنه حذفی که میکنیم ارور برطرف نشه
1- مثلا به نان ترمینال A رسیدیم ولی به ازاش حرکتی نداریم پس به ارور می خوریم و اگر به این ارور برسیم میشه ورودی رو اسکیپ کرد تا جایی که برسیم به یه چیزی که توی follow(A است
2- از ورودی اینقدر اسکیپ می کنیم تا برسیم به یه چیزی که توی firsrt(A است --> A توی استک است ولی براش حرکتی نداریم
3- اگر یک نان ترمینالی بتونه اپسیلون رو تولید کنه و به یه جایی بخوریم که دچار ارور بشیم می تونیم اون نان ترمینال رو حذف کنیم به این صورت که به جاش اپسیلون بذاریم
4- اگر دوتا ترمینال مثل هم بود یک حرف رو نادیده می گیره و می ره حرف بعدی مثلا توی استک id بود و توی ورودی + و می ره حرف بعدی توی استک رو می بینه --> پس ترمینال رو پاک می کنن از روی استک

Error Recovery in Predictive Parsing

- **Panic Mode**
  5. Often, there is a hierarchical structure on constructs in a language; for example, expressions appear within statements, which appear within blocks, and so on. We can add to the synchronizing set of a lower-level construct the symbols that begin higher-level constructs. For example, we might add keywords that begin statements to the synchronizing sets for the nonterminals generating expressions

- **Example**

NON - TERMINAL	INPUT SYMBOL					
	id	$+$	$*$	(	)	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E \rightarrow +TE'$			$E \rightarrow \epsilon$	$E \rightarrow \epsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow$ **id**	synch	synch	$F \rightarrow (E)$	synch	synch

-
5- این وابسته به کامپایلرها و زبان های برنامه نویسی است

توی جدول برای حالت های synch ارور ریکاوری تعریف شده ولی برای جاهای خالی به خطا می خوره

برای E و * چون ارور ریکاوری تعریف نکردیم و اگر به این مورد خورد خطا میده بهش و میاد بیرون

Error Recovery in Predictive Parsing

- **Example**

STACK	INPUT	REMARK
$E\ \$$	$)\ \mathbf{id} * + \mathbf{id}\ \$$	error, skip)
$E\ \$$	$\mathbf{id} * + \mathbf{id}\ \$$	**id** is in FIRST(E)
$TE'\ \$$	$\mathbf{id} * + \mathbf{id}\ \$$	
$FT'E'\ \$$	$\mathbf{id} * + \mathbf{id}\ \$$	
$\mathbf{id}\ T'E'\$$	$\mathbf{id} * + \mathbf{id}\ \$$	
$T'E'\ \$$	$* + \mathbf{id}\ \$$	
$* FT'E'\ \$$	$* + \mathbf{id}\ \$$	
$FT'E'\ \$$	$+ \mathbf{id}\ \$$	error, $M[F,+] = \text{synch}$ F has been popped
$T'E'\ \$$	$+ \mathbf{id}\ \$$	
$E'\ \$$	$+ \mathbf{id}\ \$$	
$+ TE'\ \$$	$+ \mathbf{id}\ \$$	
$TE'\ \$$	$\mathbf{id}\ \$$	
$FT'E'\ \$$	$\mathbf{id}\ \$$	
$\mathbf{id}\ T'E'\ \$$	$\mathbf{id}\ \$$	
$T'E'\ \$$	$\$$	
$E'\ \$$	$\$$	
$\$$	$\$$	

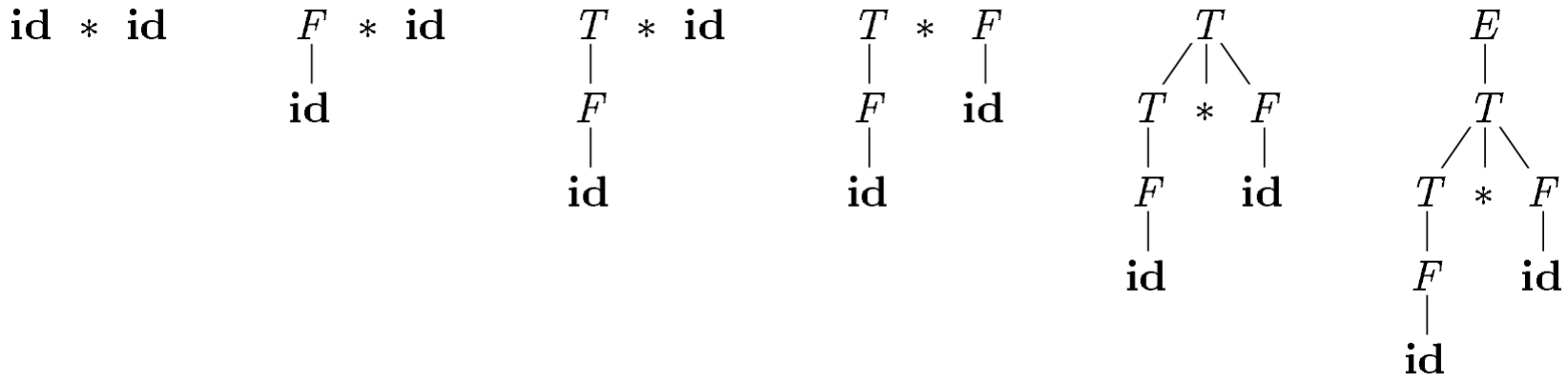
به کاربر ارور میده ولی اسکیپ هم میکنه که بتونه ادامه بده

از توی استک F رو پاپ کرده

اینا خودش توی کامپایلر تعریف شده که چی حذف بشه و چی پاپ بشه و چطوری بره جلو

Bottom-Up Parsing

- A bottom-up parse corresponds to the construction of a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top)
- **Example**



-

اینجا هم از سمت چپ شروع میکنیم به خوندن : L اول
L دوم: اشتقاق --> سمت راست ترین اشتقاق میشه

Bottom-Up Parsing

- **Reductions**
  - We can think of bottom-up parsing as the process of ***reducing*** a string w to the start symbol of the grammar
  - The key decisions during bottom-up parsing are about when to reduce and about what production to apply, as the parse proceeds
  - **Example**
    - The sequence of reductions in the previous example:
      - $id * id,\ F * id,\ T * id,\ T * F,\ T,\ E$
  - A reduction is the reverse of a step in a derivation
  - The goal of bottom-up parsing is therefore to construct a derivation in reverse
  - **In previous example:** $E \Rightarrow T \Rightarrow T * F \Rightarrow T * \mathbf{id} \Rightarrow F * \mathbf{id} \Rightarrow \mathbf{id} * \mathbf{id}$
    - ***This derivation is in fact a rightmost derivation***

Bottom-Up Parsing

- **Handle Pruning**
  - Bottom-up parsing during a left-to-right scan of the input constructs a ***rightmost derivation in reverse***
  - A ***handle*** is a substring that matches the body of a production, and whose reduction represents one step along the reverse of a rightmost derivation
  - **Example**

RIGHT SENTENTIAL FORM	HANDLE	REDUCING PRODUCTION
$\mathbf{id}_1 * \mathbf{id}_2$	$\mathbf{id}_1$	$F \rightarrow \mathbf{id}$
$F * \mathbf{id}_2$	F	$T \rightarrow F$
$T * \mathbf{id}_2$	$\mathbf{id}_2$	$F \rightarrow \mathbf{id}$
$T * F$	$T * F$	$T \rightarrow T * F$
T	T	$E \rightarrow T$

-

سمت راست قاعده است --> هندل: باید سمت راست یک قاعده دیده بشه

Bottom-Up Parsing

- **Shift-Reduce Parsing**
  - **Shift-reduce parsing is a form of bottom-up parsing** in which a stack holds grammar symbols and an input buffer holds the rest of the string to be parsed
  - The **handle always appears at the top of the stack** just before it is identified as the handle
  - We use $ to mark the bottom of the stack and also the right end of the input
  - In bottom-up parsing, we show **the top of the stack on the right**, rather than on the left as we did for top-down parsing

---

Bottom-Up Parsing

- **Shift-Reduce Parsing**
  - Initially, the stack is empty, and the string w is on the input

STACK	INPUT
$	w $

  - If parser enters the following configuration announces successful completion of parsing

STACK	INPUT
$ S	$

-
پایین به بالا: توی جدوله یا شیفت است یا کاهش --> یکی از این دوتا می تونه باشه فقط
کاهش:
شیفت:
$ ب انتهای رشته اضافه میکنه و توی استک هم $ رو می ذاریم

Bottom-Up Parsing

- **Shift-Reduce Parsing**
  - **Example**

STACK	INPUT	ACTION
$	$\mathbf{id}_1 * \mathbf{id}_2$ $	shift
$ $\mathbf{id}_1$	$* \mathbf{id}_2$ $	reduce by $F \rightarrow \mathbf{id}$
$ F	$* \mathbf{id}_2$ $	reduce by $T \rightarrow F$
$ T	$* \mathbf{id}_2$ $	shift
$ $T *$	$\mathbf{id}_2$ $	shift
$ $T * \mathbf{id}_2$	$	reduce by $F \rightarrow \mathbf{id}$
$ $T * F$	$	reduce by $T \rightarrow T * F$
$ T	$	reduce by $E \rightarrow T$
$ E	$	accept

حالت پذیرش:
توی ورودی کامل تموم بشه ولی توی استک یه چیزی باشه

$ ته استک است به شکل خوب نگاه کن

شیفت: ینی از ورودی به داخل استک ینی یه چیزی از ورودی برداشته بشه و بیاد توی استک این میشه شیفت
کاهش: اون چیزی که بالای استک است میشه هندل توی گرامر می تونیم اینجا کاهش انجام بدیم پس کاهش برای اینکه بخواد ممکن بشه باید یک هندل بالای استک داشته باشیم --> الان وضعیت شروع همین است هیچی توی استک نیست پس کاهش ممکن نیست --> توی استک ما پاپ و پوش داریم کلا
پذیرش:
ارور: ینی توی وضعیت پذیرش نباشیم

F می تونه هندل باشه و به جاش T بیاد !!!

نکته: هندل همیشه ظاهر میشه در بالای استک

Bottom-Up Parsing

- **Shift-Reduce Parsing**
  - Possible actions a shift-reduce parser can make
    1. **Shift**
       - Shift the next input symbol onto the top of the stack
    2. **Reduce**
       - The right end of the string to be reduced must be at the top of the stack
       - Locate the left end of the string within the stack and decide with what nonterminal to replace the string
    3. **Accept**
       - Announce successful completion of parsing.
    4. **Error**
       - Discover a syntax error and call an error recovery routine.

Bottom-Up Parsing

- Conflicts During Shift-Reduce Parsing
  - **Shift/Reduce conflict**

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if}\ expr\ \textbf{then}\ stmt \\
 & | & \textbf{if}\ expr\ \textbf{then}\ stmt\ \textbf{else}\ stmt \\
 & | & \textbf{other}
\end{array}
$$

STACK	INPUT
$\cdots$ **if** *expr* **then** *stmt*	**else** $\cdots$ \$

  - **Reduce/Reduce conflict**

$$
\begin{array}{rcl}
(1) \quad stmt & \rightarrow & \textbf{id}\ (\ parameter_list\ ) \\
(2) \quad stmt & \rightarrow & expr\ :=\ expr \\
(3) \quad parameter_list & \rightarrow & parameter_list\ ,\ parameter \\
(4) \quad parameter_list & \rightarrow & parameter \\
(5) \quad parameter & \rightarrow & \boxed{\textbf{id}} \\
(6) \quad expr & \rightarrow & \textbf{id}\ (\ expr_list\ ) \\
(7) \quad expr & \rightarrow & \boxed{\textbf{id}} \\
(8) \quad expr_list & \rightarrow & expr_list\ ,\ expr \\
(9) \quad expr_list & \rightarrow & expr
\end{array}
$$

STACK	INPUT
$\cdots$ **id** (**id**	, **id**) $\cdots$

-

هم می تونیم if رو کاهش بدیم و هم می تونیم else رو شیفت بدیم --> اینجا هم میشه کاهش رو انجام داد و هم شیفت رو انجام داد پس اینجا تصادم داریم : اولی