Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

$LL(1)$ Grammars

- **$LL(1)$**
  - The first "L" in $LL(1)$ stands for scanning the input from left to right
  - The second "L" for producing a leftmost derivation
  - The "1" for using one input symbol of lookahead at each step to make parsing action decisions
- **A grammar G is $LL(1)$ if and only if whenever $A \to \alpha|\beta$ are two distinct productions of G, the following conditions hold:**
  1. $FIRST(\alpha)$ and $FIRST(\beta)$ are disjoint sets
  2. If ϵ is in $FIRST(\beta)$, then $FIRST(\alpha)$ and $FOLLOW(A)$ are disjoint sets, and likewise if ϵ is in $FIRST(\alpha)$
- Predictive parsers can be constructed for $LL(1)$ grammars

LL(1) Grammars

- **Construction of a predictive parsing table**

INPUT: Grammar G.

OUTPUT: Parsing table M.

METHOD: For each production $A \to \alpha$ of the grammar, do the following:

1. For each terminal a in FIRST(α), add $A \to \alpha$ to $M[A, a]$.
2. If ϵ is in FIRST(α), then for each terminal b in FOLLOW(A), add $A \to \alpha$ to $M[A, b]$. If ϵ is in FIRST(α) and \$ is in FOLLOW(A), add $A \to \alpha$ to $M[A, \$]$ as well.

- If, after performing the above, there is no production at all in $M[A, a]$, then set $M[A, a]$ to error

-
گرامرهای LL میشن گرامرهای بالا به پایین
گرامرهای LR میشن گرامرهای پایین به بالا که بعدا می گیم اینو

یک گرامر 1)LL) اگر و فقط اگر:
1- فرست الفا و فرست بتا نباید اشتراک داشته باشن ؟؟
2- اگر اپسیلون رو توی فرست بتا داشته باشیم؟

نفهمیدم این صفحه رو؟؟؟

LL(1) Grammars

- **Example**

@

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

جدول pars:

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

وقتی که میخوایم LL(1) بودن یک گرامر رو تشخیص بدیم یک همچین جدولی براش می سازیم و پر میکنیم و اگر تو این جدول که ساخته میشه هیچ خونه ای بیشتر از یک قاعده داخلش نباشه این میشه LL(1)
الان این جدوله LL(1) است
حالا این جدول چطوری ساخته میشه؟
سطرهاش میشه اسم متغییرها ینی نان ترمینال ها
و ستون هاش میشه اسم ترمینال ها
$ هم انتهای رشته اضافه میکنیم که مطمئن بشیم اون عملیات پارسر تموم شده و رسیدیم به انتهای رشته
برای هر یه دونه قاعده ای از گرامر: یه قاعده رو انتخاب میکنیم و میخوایم ببینیم توی کدوم سلول های جدول می تونیم این قاعده رو بنویسیم --> برای هر قاعده میگیم هر ترمینالی که توی فرست سمت راست اون قاعده باشه باید توی سلول مربوط به که سطرش مثلا A است و ستونش اون ترمیناله است باید بیاد مثلا برای قاعده @ فرست این قاعده میشه علامت + پس باید توی سطر مربوط به E' و ستون + این قاعده بیاد پس برای هر قاعده در ابتدا می بینیم فرستش چی هست و برای اون فرست میایم قاعده مربوطه رو می نویسیم --> اینجا هم به فرست کار داریم و هم به فالو به این صورت است که اول برای هر قاعده ای فرستش رو می بینیم و اگر فرستش اپسیلون نداشت فقط باید بیایم اون قاعده رو توی ستون های مربوط به فرست ها بذاریم ولی اگر توی فرستش اپسیلون داشت باید فالو رو ببینیم و برای همه ستون هایی که توی اون فالو هستن باید اون قاعده رو بنویسیم مثلا برای E' یکی از فرست هاش میشه اپسیلون که توی این حالت می ریم سراغ فالو E' و برای همه ستون هایی که توی اون فالو هستن قاعده رو می نویسیم
نکته: با این سلول هایی که خالی می مونن، کامپایلرها به صورت های مختلفی باهاشون برخورد می کنن و معمولا این ها حالت ارور است مثلا اگر رسیدیم به استیت E و جمع رو ببینیم هیچ حالتی نداریم که بخوایم بهش حرکت کنیم پس این یک حالت ارور است برای کامپایلر که باید ارور رو بده به کاربر

LL(1) Grammars

- For every *LL*(1) grammar, each parsing-table entry uniquely identifies a production or an error
- **Example**

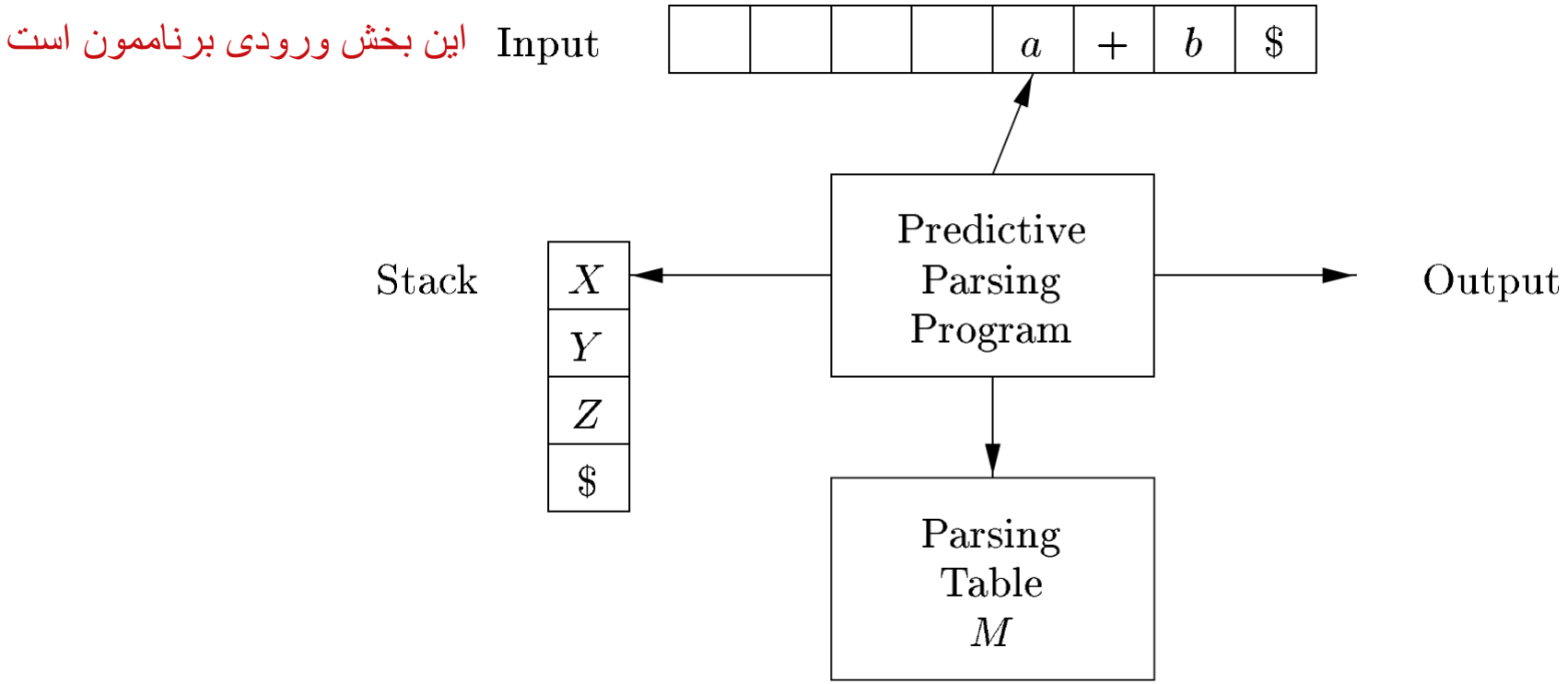
$$
\begin{aligned}
S &\rightarrow iEtSS' \mid a \\
S' &\rightarrow eS \mid \epsilon \\
E &\rightarrow b
\end{aligned}
$$

Non-terminal	Input Symbol					
	a	b	e	i	t	\$
S	$S \rightarrow a$			$S \rightarrow iEtSS'$		
S'			$S' \rightarrow \epsilon$ $S' \rightarrow eS$			$S' \rightarrow \epsilon$
E		$E \rightarrow b$				

-
در این جدول توی یک خونه + بیشتر از یک قاعده است پس LL(1) نیست

Nonrecursive Predictive Parsing

- A nonrecursive predictive parser can be built by maintaining a stack explicitly, rather than implicitly via recursive calls



Nonrecursive Predictive Parsing

- **Table-driven predictive parsing**
  - Initially, $w\$$ in the input buffer and the start symbol S of G on top of the stack, above \$

```
let a be the first symbol of w;
let X be the top stack symbol;
while ( X ≠ $ ) { /* stack is not empty */
        if ( X = a ) pop the stack and let a be the next symbol of w;
        else if ( X is a terminal ) error();
        else if ( M[X, a] is an error entry ) error();
        else if ( M[X, a] = X → Y1 Y2 ··· Yk ) {
                output the production X → Y1 Y2 ··· Yk;
                pop the stack;
                push Yk, Yk−1, . . . , Y1 onto the stack, with Y1 on top;
        }
        let X be the top stack symbol;
}
```

اینجا id با id مچ شده پس اون از ورودی حذف میشه و اون یکی هم از پشته پاپ میشه

به انتهای رشته $ اضافه میشه

T بالای استک است یعنی اول E' پوش شده و بعد T چون T باید بالای استک قرار بگیره

- **Example**

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E' \mid \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T' \mid \epsilon \\
F & \rightarrow & (\ E\ ) \mid \mathbf{id}
\end{array}
$$

MATCHED	STACK	INPUT	ACTION
	$E\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	
	$TE'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $E \rightarrow TE'$
	$FT'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $T \rightarrow FT'$
	$\mathbf{id}\ T'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $F \rightarrow \mathbf{id}$
$\mathbf{id}$	$T'E'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	match $\mathbf{id}$
$\mathbf{id}$	$E'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	output $T' \rightarrow \epsilon$
$\mathbf{id}$	$+\ TE'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	output $E' \rightarrow +\ TE'$
$\mathbf{id} +$	$TE'\$$	$\mathbf{id} * \mathbf{id}\$$	match $+$
$\mathbf{id} +$	$FT'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $T \rightarrow FT'$
$\mathbf{id} +$	$\mathbf{id}\ T'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $F \rightarrow \mathbf{id}$
$\mathbf{id} + \mathbf{id}$	$T'E'\$$	$*\ \mathbf{id}\$$	match $\mathbf{id}$
$\mathbf{id} + \mathbf{id}$	$*\ FT'E'\$$	$*\ \mathbf{id}\$$	output $T' \rightarrow *\ FT'$
$\mathbf{id} + \mathbf{id} *$	$FT'E'\$$	$\mathbf{id}\$$	match $*$
$\mathbf{id} + \mathbf{id} *$	$\mathbf{id}\ T'E'\$$	$\mathbf{id}\$$	output $F \rightarrow \mathbf{id}$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$T'E'\$$	$\$$	match $\mathbf{id}$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$E'\$$	$\$$	output $T' \rightarrow \epsilon$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$\$$	$\$$	output $E' \rightarrow \epsilon$

این حالت پذیرش است

می خوایم یک ورودی رو بررسی کنیم و ببینیم توسط این گرامر تولید میشه یا نمیشه

اون چیزی که در نهایت باید بهش برسیم اگر بخوایم این ورودی پذیرش بشه توسط این گرامر حالت + است ینی استک خالی بشه و برسه به $ و ورودی هم تموم بشه و برسه به $ --> اگر همزمان ورودی تموم شد و استک هم خالی شد این پذیرش میشه و اگر هر حالت دیگه ای پیش بیاد این پذیرش نمیشه

این جدول اون کار اصلی است که تحلیل گر نحوی داره انجام میده چون تحلیل گر نحوی قراره برنامه رو بگیره و بگه گرامرش درست است یا نه

نکته: این الگوریتم طبق این پیش می ره که استک خالی بشه پس تا زمانی که استک خالی بشه پیش می ریم و اگر به جایی برسیم که استک خالی نشه این ورودی پذیرش نمیشه