Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

Verifying the Language Generated by a Grammar

- A proof that a grammar G generates a language L has two parts:
  - Show that every string generated by G is in L
  - Show that every string in L can indeed be generated by G
- **Example**
  - **The following grammar generates all strings of balanced parentheses**

$$S \rightarrow (\ S \ ) \ S \ \mid \ \epsilon$$

رشته های گرامر بالا: رشته هایی که کلا از پرانتز باز و پرانتز بسته تشکیل شده اند --> پرانتز باز و بسته اش بالانس است

-
اگر بخوایم نشان بدیم گرامر G و زبان L با هم معادل اند --> هر رشته ای که توسط G تولید میشه توی L است و هر رشته ای که توی L باشه توسط G تولید میشه

Context-Free Grammars Versus Regular Expressions

- **Every regular language is a context-free language, but not vice-versa**
- **Example**
  - Construct a context-free grammar from an NFA

$$(a|b)^*abb$$

$$\begin{array}{lcl} A_0 & \rightarrow & aA_0 \mid bA_0 \mid aA_1 \\ A_1 & \rightarrow & bA_2 \\ A_2 & \rightarrow & bA_3 \\ A_3 & \rightarrow & \epsilon \end{array}$$

- The language $L = \{a^n b^n \mid n \geq 1\}$ is context-free but not regular
- ***Finite automata cannot count***

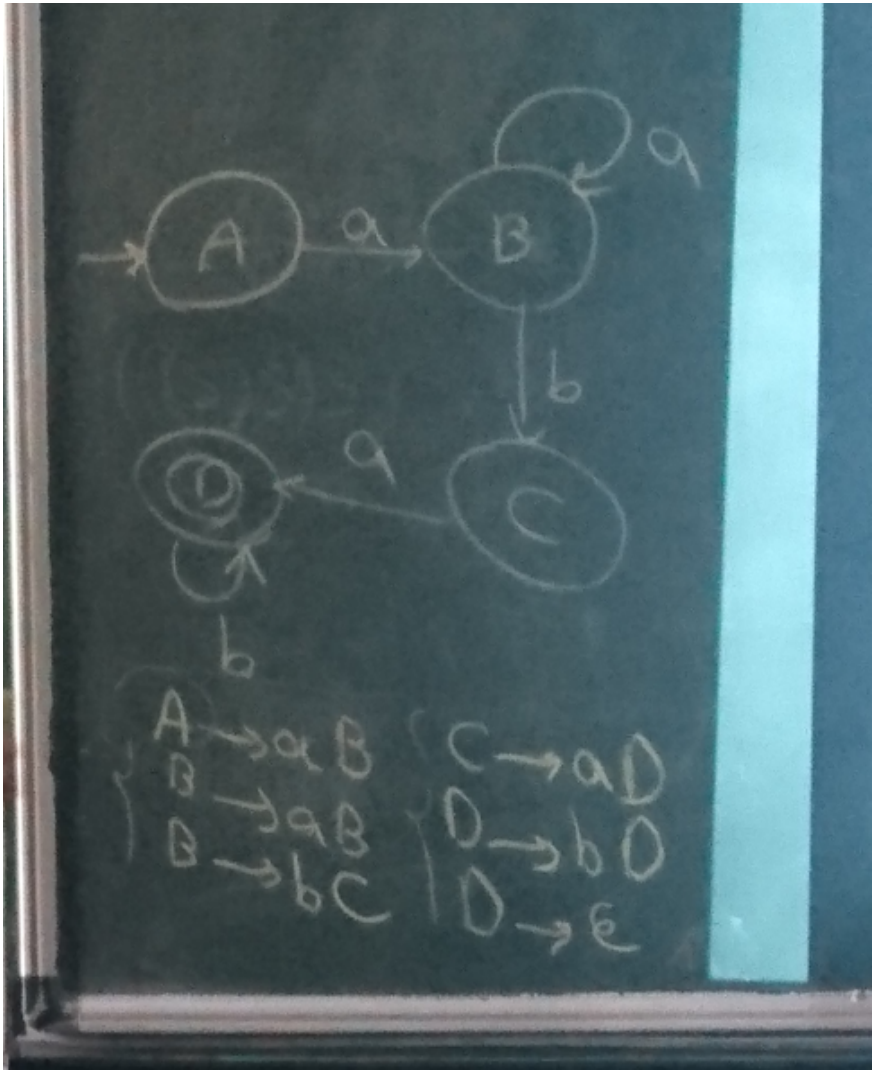
زبان مستقل از متن است ولی زبان منظم نیست

چون n رو نداریم و براش نمی تونیم DFA یا NFA رسم کنیم --> همین مساوی بودن که باید چک بشه و نمی تونیم چک بکنیم باعث میشه که این زبان منظم نباشه چون ما براش نمی تونیم DFA یا NFA رسم کنیم ولی مستقل از متن است چون که خیلی گرامر میشه براش نوشت با روش های مختلف و گرامری که میشه براش نوشت: S-> aSb|e فرض کن اینجا e همون اپسیلون است چون نداشتم که بخوام بذارم:)

-
تفاوت زبان های منظم و مستقل از متن:
هر زبان منظمی یک زبان مستقل از متن است ولی برعکسش برقرار نیست
توی زبان های منظم: اگر بخوایم از طریق دیاگرام بخوایم بگیم با یک NFA , DFA میشه نشونش داد ولی اگر از طریق عبارت منظم بخوایم بگیم که عبارت های منظم می تونست از اجتماع تشکیل شده باشه و الحاق و بستار ینی هر زبانی که با استفاده از این همین سه تا بشه نوشتش میشه منظم
ولی زبان های مستقل از متن همه منظم نیستن ینی قاعدتا زبان هایی هستن که ما نمی تونیم به صورت عبارت منظم بنویسیمشون ولی برعکسش ینی ما هر زبان منظمی داشته باشیم می تونیم یک گرامر مستقل از متن براش بنویسیم
اگر برای یک زبان منظم یک NFA , DFA رسم کنیم از روی اون خیلی راحت میشه براش گرامر نوشت

نکته: زبان های مستقل از متن نمی تونه اتوماتای متناهی که همون NFA , DFA میشه رو بشماره ینی هر چیزی که نیاز به شمردن و نگه داشتن و اینجور چیزا داشته باشه توسط زبان های مستقل از متن قابل پیاده سازی نیست

مثلا اینجا یک دیاگرام برای زبان منظم داریم حالا از روی این دیاگرام می تونیم براش گرامر بنویسم به این صورت که برای هر یال می تونیم یک قانون بنویسیم:
برای استیت پایانی می تونیم یک اپسیلون بذاریم ینی بگیم D می دهد اپسیلون



Context-Free Grammars

- **Example**

Consider the context-free grammar:

$$S \rightarrow S\,S\,+ \mid S\,S\,* \mid a$$

and the string $aa + a*$.

a) Give a leftmost derivation for the string.

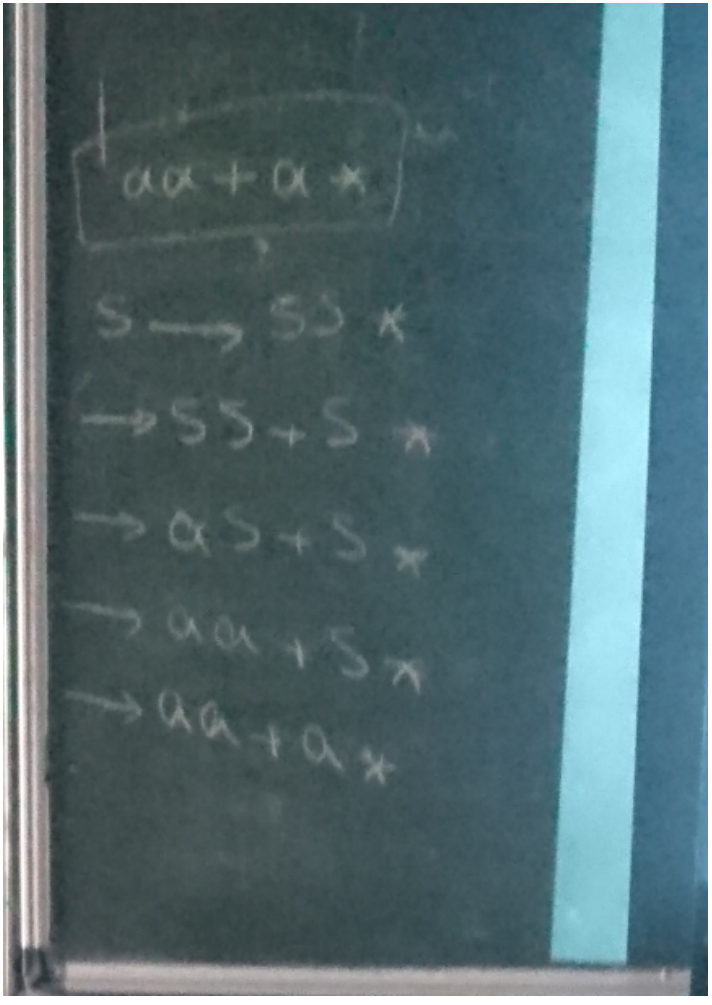
b) Give a rightmost derivation for the string.

c) Give a parse tree for the string.

! d) Is the grammar ambiguous or unambiguous? Justify your answer.

! e) Describe the language generated by this grammar.

اشتقاق سمت چپ ترین: -->
در هر مرحله فقط یک جایگزینی انجام میدهیم



ایا این گرامر مبهم است؟ ینی ایا می تونیم یک اشتقاق دیگه ای بنویسیم که باز این سمت چپ ترین رو تولید بکنه؟ نه

زبانی هم که تولید میکنه میشه چه زبانی؟
حالت عملگر بعد از عملوندها اومده پس میشه حالت post fix

Elimination of Left Recursion

- A grammar is left recursive if it has a nonterminal A such that there is a derivation $A \overset{+}{\Rightarrow} A\alpha$ for some string α قاعده بازگشتی سمت چپ:
- Top-down parsing methods cannot handle left-recursive grammars, so a transformation is needed to eliminate left recursion
- **Example**

2 تا بازگشتی سمت چپ اینجا هست

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T * F \mid F \\ F & \rightarrow & (E) \mid \mathbf{id} \end{array}$$

++

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

- **Example**

قاعده کلی

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

$$\begin{array}{l} A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{array}$$

-
در نهایت می خوایم به این برسیم که گرامر پایین به بالا و بالا به پایین چجوری پیاده سازی میشه که اصل بحث تحلیل نحوی است
قبل از اینکه به اون بالایی برسیم یکسری گرامرهایی هستن مثل همین مبهم که باعث میشه مشکل پیش بیاد در پیاده سازی و معمولا سعی می کنند به یک طریقی گرامرهای مبهم رو حذف کنن تا بتونند راحت پیاده سازی بکنند
غیر از اینا ما دو نوع گرامر دیگه هم داریم که اینا هم معمولا توی پیاده سازی مخصوصا به روش بالا به پایین مشکل پیش میارن --> پس به این دو سبکی که میگیم اگه داشتیم سعی می کنیم اینا هم حذف بشن یا از این حالت دربیان
1- حذف کردن بازگشتی سمت چپ --> بازگشتی سمت چپ ینی خود متغییر سمت چپ باشه که اینجا A هست
سمت راست وقتی که بازگشتی باشه معمولا توی پیاده سازی ها مشکلی نداره ولی بازگشتی سمت چپ معمولا مشکل پیش میاره و اینو سعی میکنن حذف بکنن و یکسری جایگزین هایی براش بنویسن که توی این حالت نباشه و بعد پیاده سازی اون درخت رو انجام بدن
:++
گرامر معادلی می نویسیم که این بازگشتی سمت چپ رو نداره
روشش این است که:
اول از همه توی این قاعده ای که هست ینی اینجا سمت راستش دوتا قاعده است میایم اول اون قاعده یا قاعده هایی که E با انها تمام میشه که اینجا T است --> حالا اون عبارتی که بهش می رسیم رو می ذاریم اول و بعد یک متغییر جدید دیگه تعریف می کنیم
در گام دوم که E' داریم این E' باید باشه چون حالت بازگشتی رو نمی خوایم از بین ببریم ولی میخوایم ببریمش سمت راست قرارش بدیم چون سمت راست توی پیاده سازی مشکلی نداره ولی سمت چپ هست که توی پیاده سازی مشکل به وجود میاره

Elimination of Left Recursion

- **Algorithm to eliminate left recursion from a grammar**

1) arrange the nonterminals in some order $A_1, A_2, \ldots, A_n$.
2) **for** (each i from 1 to n) {
3) **for** (each j from 1 to $i-1$) {
4) replace each production of the form $A_i \rightarrow A_j\gamma$ by the productions $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$, where $A_j \rightarrow \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ are all current A_j-productions
5) }
6) eliminate the immediate left recursion among the A_i-productions
7) }

Elimination of Left Recursion

- **Example**

اینجا به طور غیر مستقیم بازگشتی سمت چپ داریم: رنگ سبز

$$S \rightarrow A\ a \mid b$$
$$A \rightarrow A\ c \mid S\ d \mid \epsilon$$

$$A \rightarrow A\ c \mid A\ a\ d \mid b\ d \mid \epsilon$$

$$S \rightarrow A\ a \mid b$$
$$A \rightarrow b\ d\ A' \mid A'$$
$$A' \rightarrow c\ A' \mid a\ d\ A' \mid \epsilon$$

اونایی که با A شروع نمی شن رو در نظر میگیریم که bd و اپسیلون است و A' هم بخاطر اپسیلون نوشتیم

Left Factoring

- Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive, or top-down, parsing

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

- **Example**

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \quad \Rightarrow \quad \begin{array}{l} A \rightarrow \alpha A' \\ A' \rightarrow \beta_1 \mid \beta_2 \end{array}$$

-
فاکتورگیری سمت چپ:
منظور برای حالت هایی است که ما سمت چپ عبارت هامون، اون عبارتی که باهاش شروع میشن مشترک باشه مثل الفا توی مثال --> ینی سمت چپشون با یک عبارت مشترک شروع شده
نکته: قبل از پیاده سازی گرامر بالا به پایین اگر همچین چیزی هم داشتیم باید اول درستش بکنیم و بعد بریم سراغ پیاده سازی دقیقا مثل حرفی که برای بازگشتی سمت چپ زدیم

مثال:
adB مشترک است و D هم متغییر جدید است