Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

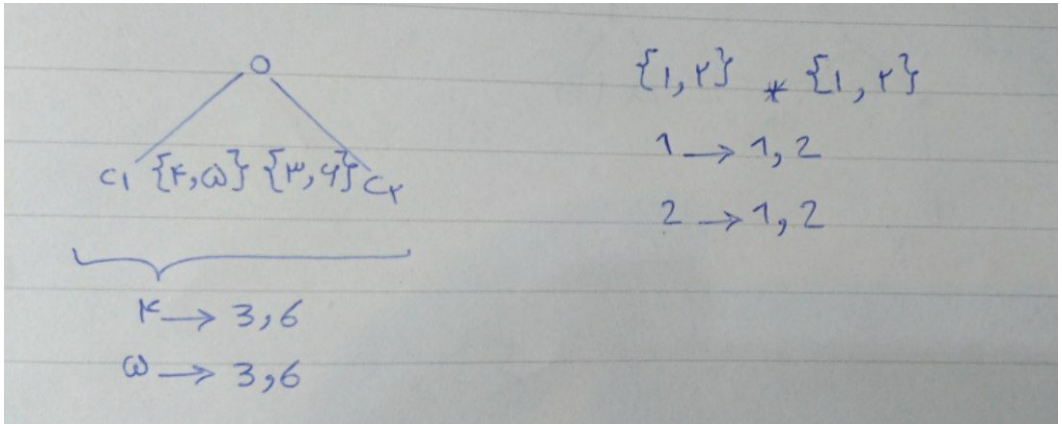
Converting a Regular Expression Directly to a DFA

- **Computing followpos**
  1. If n is a cat-node with left child c_1 and right child c_2, then for every position i in $lastpos(c_1)$, all positions in $firstpos(c_2)$ are in $followpos(i)$
  2. If n is a star-node, and i is a position in $lastpos(n)$, then all positions in $firstpos(n)$ are in $followpos(i)$

- **Example**

POSITION n	$followpos(n)$
1	$\{1, 2, 3\}$
2	$\{1, 2, 3\}$
3	$\{4\}$
4	$\{5\}$
5	$\{6\}$
6	$\emptyset$

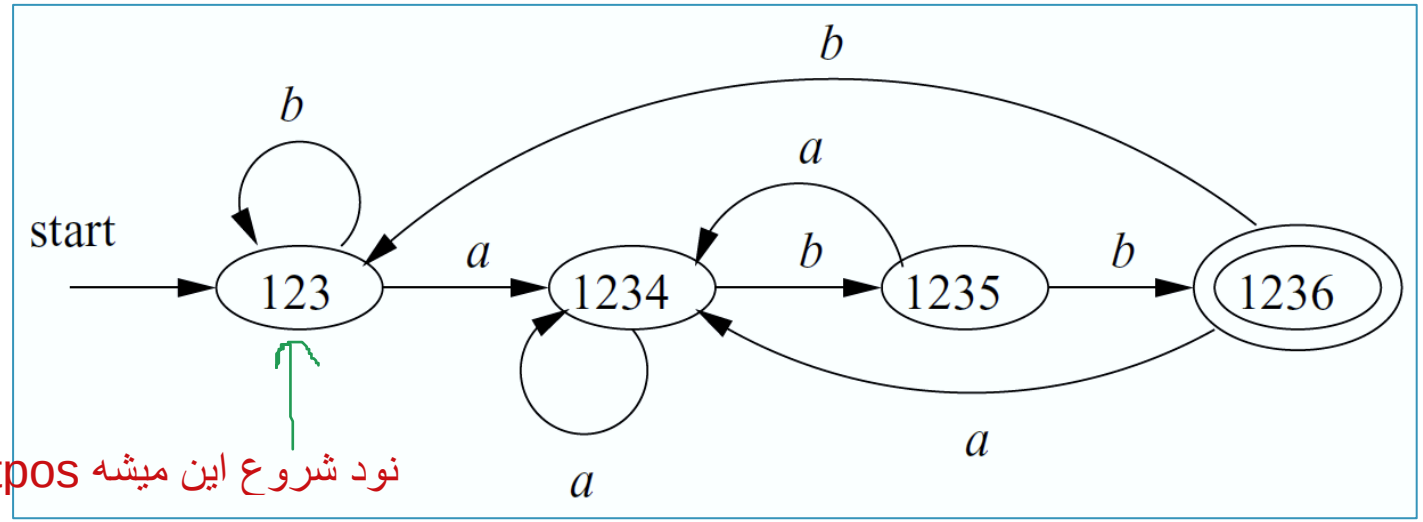
-
followpos رو ما به ازای هر مکانی یا پوزیشنی تعریف می کنیم
برای followpos دوتا قاعده داریم:
اولش فرض میکنیم که برای تمامی پوزیشن ها followpos هاشون تهی است و می خوایم یکی یکی followpos رو برای پوزیشن ها حساب بکنیم
1- روی همه نودهای درخت حرکت میکنیم و برای نودهایی که الحاق هستن و نودهایی که بستار هستن --> برای این دو نوع نود اینا می تونن followpos های مختلف رو تحت تاثیر قرار بدن
اگر یک نود الحاق داشته باشیم که زیر درخت سمت چپ ان نود C1 باشه و سمت راستش C2: همه اونایی که توی lastpos مربوط به C1 قرار دارن میشن پوزیشن های ما و مقدار این پوزیشن ها با firstpos مربوط به C2 پر میشه مثلا توی lastpos مربوط به C1 ما 4 و 5 داریم و برای firstpos مربوط به C2 ما 3 و 6 داریم حالا برای پوزیش 4 ما 3 و 6 داریم و برای پوزیش 5 هم 3 و 6 داریم
2- اگر یک بستار نود داشتیم برای همه پوزیشن هایی که توی lastpos هستن مقدارشون میشه اونایی که توی firstpos قرار دارن



Converting a Regular Expression Directly to a DFA

- **Example**

حالت پایانی هم میشه همه اون وضعیت هایی که این # که پوزیشن 6 است داخلشون باشه

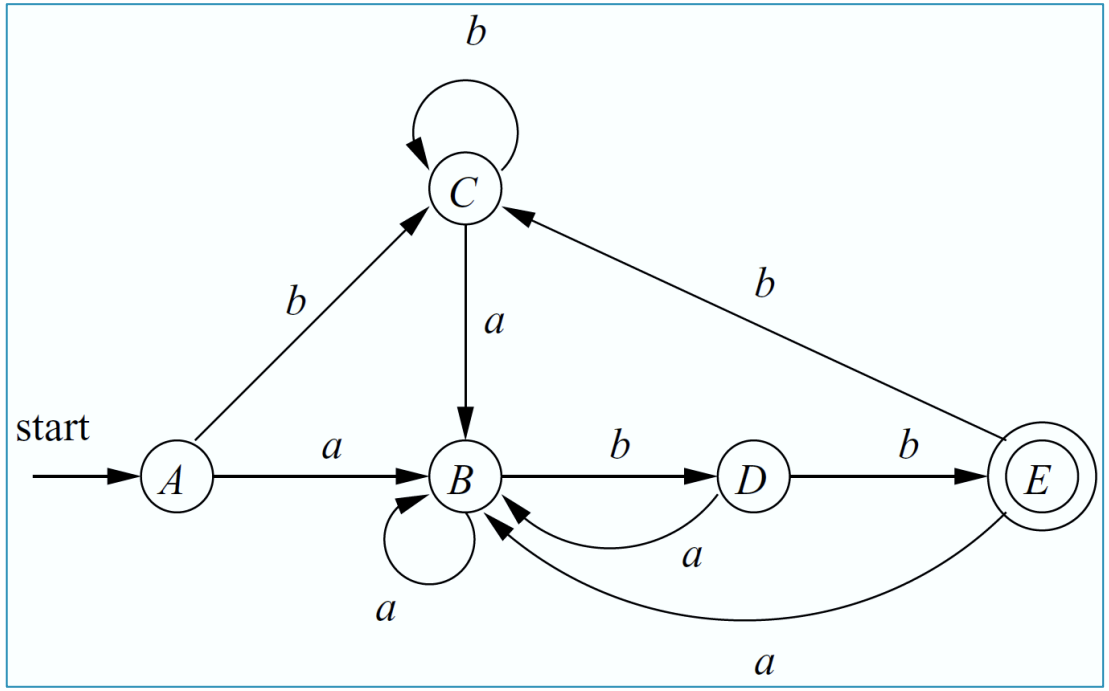


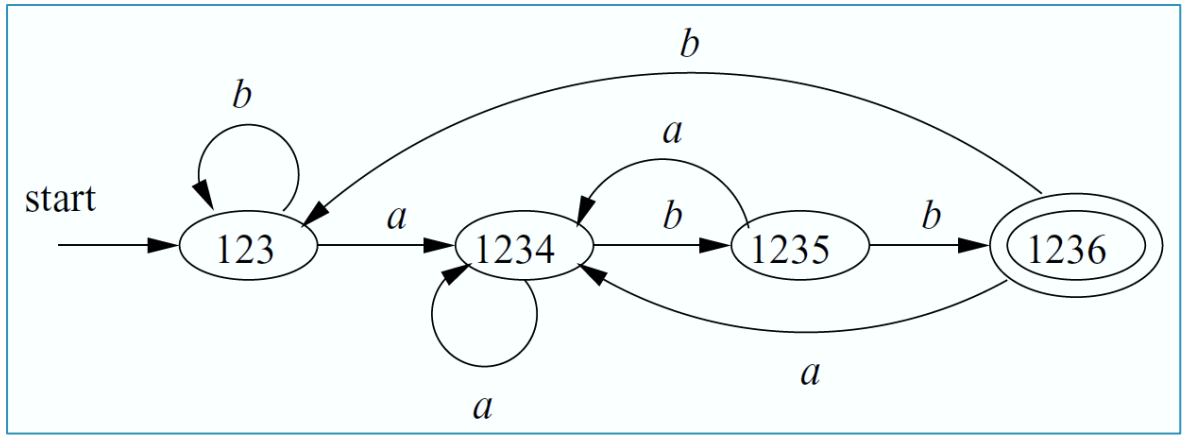
نود شروع این میشه firstpos روت درخت

حالا می بینیم از بین پوزیشن های 1 و 2 و 3 کدومشون a بوده که 1 و 3 a بودن و followpos یک و سه چند بوده ؟ {1,2,3} و {4} که اجتماع این دوتا میشه وضعیتی که با a بهش حرکت می کنیم و به همین صورت برای بقیه می ریم جلو

Minimizing the Number of States of a DFA

- There can be many DFAs that recognize the same language
- **Example:** $L((a|b)^*abb)$





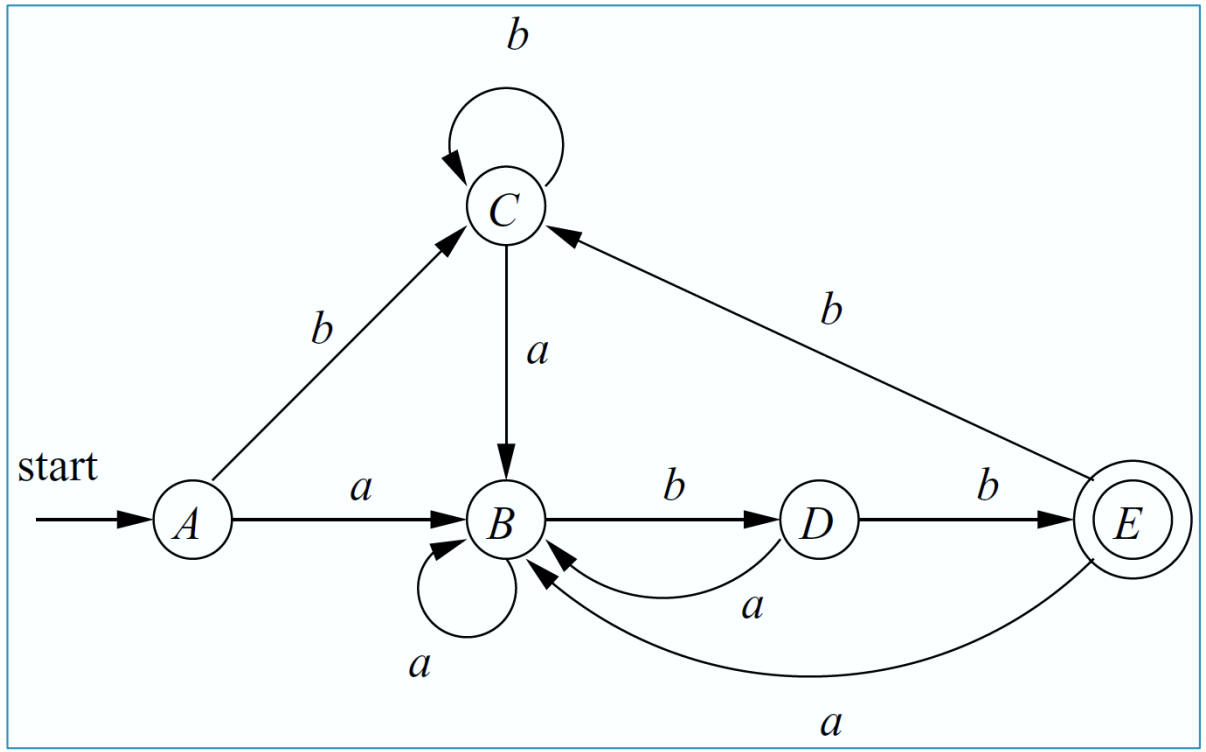
Minimizing the Number of States of a DFA

- There is always a unique minimum-state DFA for any regular language
- **Algorithm**
  1. Start with an initial partition Π with two groups, F and $S - F$, the accepting and nonaccepting states of D
  2. initially, let $\Pi_{\text{new}} = \Pi$;
     for (each group G of Π) {
     partition G into subgroups such that two states s and t
     are in the same subgroup if and only if for all
     input symbols a, states s and t have transitions on a
     to states in the same group of Π;
     /* at worst, a state will be in a subgroup by itself */
     replace G in Π_{new} by the set of all subgroups formed;
     }
  3. If $\Pi_{new}! = \Pi$, repeat step (2) with Π_{new} in place of Π

-
مینیمم کردن تعداد حالت های DFA:
حذف کردن حالت های اضافی توی DFA

Minimizing the Number of States of a DFA

- **Example**
  1. $\{A, B, C, D\}\{E\}$
  2. $\{A, B, C\}\{D\}\{E\}$
  3. $\{A, C\}\{B\}\{D\}\{E\}$



درخت نهایی چجوری کشیده میشه؟؟؟ و ایا همین گفته ها درسته؟؟؟

اول از همه دوتا مجموعه در نظر میگیریم --> همه حالت هایی که نهایی هستن رو می ذاریم توی یک مجموعه و همه حالت هایی هم که نهایی نیستن هم می ذاریم توی یک مجموعه دیگه مثل گام 1

بعد شروع میکنیم تا حدی که امکان داشته باشه شکستن مجموعه بزرگه --> شکستن چجوری انجام میشه؟ هر کدوم از این دوتا مجموعه رو در نظر میگیریم و خروجی هایی که همه اعضاش داره به ازای a به دست بیاد مثلا خود A به ازای a می ره به B و به ازای b می ره به C اگر این دوتا حرکتی که داره با a , b انجام میده و به اون وضعیت ها می ره توی خود این مجموعه باشن مشکلی نداره و A می تونه بمونه
حالا برای B نگاه میکنیم به ازای a می ره به B و به ازای b می ره به D پس B , D توی همین مجموعه هستن پس مشکلی نداره و B می تونه بمونه
برای C --> به ازای a می ره به B و به ازای b می ره به C پس B , C توی همین مجموعه هستن پس مشکلی نداره و C می تونه بمونه
برای D به ازای a می ره به B و به ازای b می ره به E و E توی این مجموعه نیست پس D رو جدا میکنیم --> گام 2

و همین کار باید ادامه پیدا کنه --> گام 3

فک کنم اینایی که گفت درست نیس؟؟؟

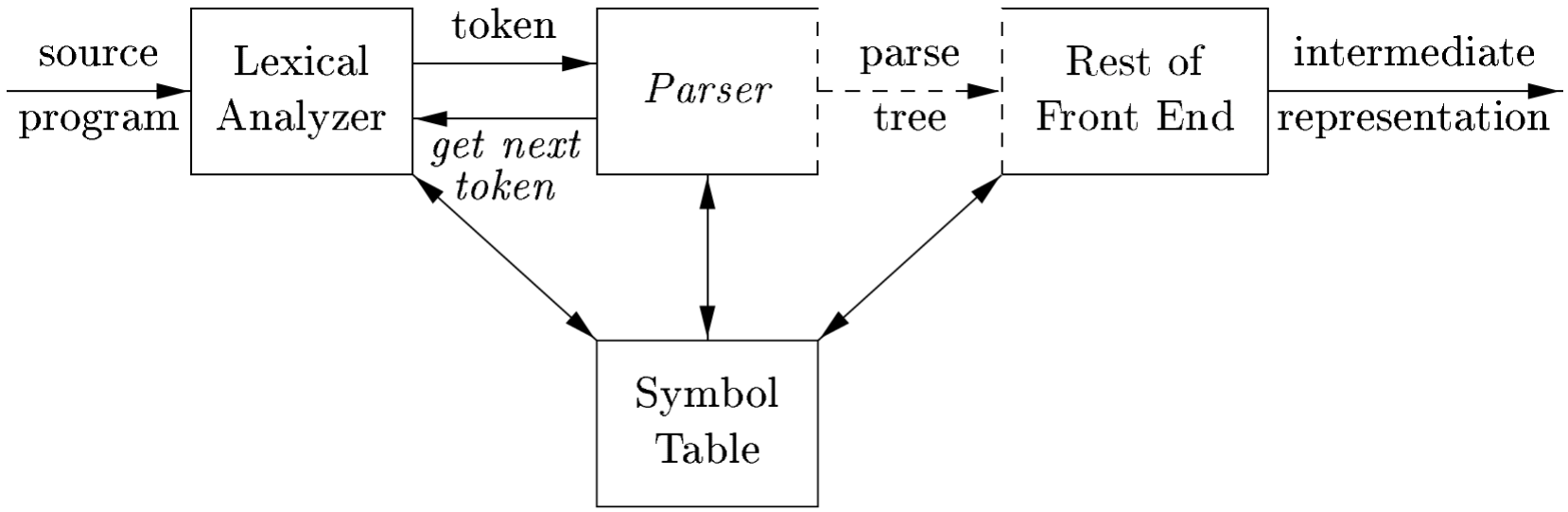
Syntax Analysis

Introduction

- Every programming language has precise rules that prescribe the syntactic structure of well-formed programs
- The syntax of programming language constructs can be specified by **context-free grammars**
- The parser **obtains a string of tokens from the lexical analyzer** and **verifies that the string of token names can be generated by the grammar for the source language**
- The parser constructs a parse tree and passes it to the rest of the compiler for further processing

-
تحلیل گر نحوی مهمترین بخش کامپایلر است --> اکثر خطاها توی این بخش است
تحلیل گر نحوی از طریق زبان های مستقل از متن پیاده سازی میشه
همه زبان های برنامه نویسی برای خودشون یک گرامر مشخصی دارند که براساس اون گرامرها زبان بررسی میشه و تحلیل میشه که مشکلی براساس اون گرامر نداشته باشه --> این کار از طریق زبان های مستقل از متن انجام میشه توی تحلیل گر نحوی
زبان های مستقل از متن کامل می تونن تحلیل گر نحوی رو پوشش بدن و پیاده سازی کنن
گرامر مستقل از متن در مقابل گرامر حساس به متن قرار میگیره

Introduction



کاری که parser انجام میده:
یک توالی از توکن ها رو از تحلیل گر لغوی می گیره و تایید کنه که این توالی از توکن ها رو که گرفته توسط گرامرهای مربوط به زبان تولید می شه یا نمیشه و اگر گرامر درستی روش داره که اوکی است و اگر نداره باید یک خطایی رو نشون بده
برای این کار یک parse tree می سازه که این parse tree یک دیتا استراکچر اصلی است که توی بقیه قسمت ها ازش استفاده میشه ینی این parse tree که این می سازه پاس داده میشه به بقیه فازهای کامپایلر

Introduction

- The methods commonly used in compilers for parser can be classified as:
  - **Top-down**
    - Top-down methods build parse trees from the top (root) to the bottom (leaves)
  - **Bottom-up**
    - Bottom-up methods start from the leaves and work their way up to the root
- The input to the parser is scanned from left to right, one symbol at a time

-
مدل هایی که استفاده میشه برای کامپایلرها برای parser به دو دسته تقسیم میشه:
Top-down: ینی اون parse tree که قراره ساخته بشه بالا به پایین ساخته بشه ینی از ریشه ساختنش شروع بشه و به برگ برسه
Bottom-up: یا پایین به بالا ینی از برگ ساختنش شروع بشه و به ریشه برسه

Context-Free Grammars

- **A context-free grammar consists of:**
  - **Terminals**
    - Terminals are the basic symbols from which strings are formed
  - **Nonterminals**
    - Nonterminals are syntactic variables that denote sets of strings
  - **Start symbol**
    - One nonterminal is distinguished as the start symbol
  - **Productions**
    - The productions of a grammar specify the manner in which the terminals and nonterminals can be combined to form strings
    - **A production consists of:**
      - A nonterminal called the head or left side of the production
      - The symbol →
      - A body or right side consisting of zero or more terminals and nonterminals

-
گرامرهای مستقل از متن: از 4 تا بخش تشکیل شده:
Terminals: که همین حروف کوچیک هست
Nonterminals: با حروف بزرگ نشون میدیم
Start symbol: جایی که گرامر ازش شروع میشه
Productions: قواعدمون است:
برای هر قاعده یک Nonterminals میاد سمت چپش
و بعد علامت $\longrightarrow$ میاد
و بعد ترکیبی از Nonterminals و Terminals و یا اپسیلون می تونه سمت راست بیاد

Context-Free Grammars

- **Example**
  - Terminal: id + - * / ( )
  - Nonterminals: expression, term, factor

$$
\begin{array}{rcl}
\mathit{expression} & \rightarrow & \mathit{expression} + \mathit{term} \\
\mathit{expression} & \rightarrow & \mathit{expression} - \mathit{term} \\
\mathit{expression} & \rightarrow & \mathit{term} \\
\mathit{term} & \rightarrow & \mathit{term} * \mathit{factor} \\
\mathit{term} & \rightarrow & \mathit{term} \;/\; \mathit{factor} \\
\mathit{term} & \rightarrow & \mathit{factor} \\
\mathit{factor} & \rightarrow & (\; \mathit{expression} \;) \\
\mathit{factor} & \rightarrow & \mathbf{id}
\end{array}
$$

- **Example**

$$
\begin{array}{rcl}
E & \rightarrow & E + T \mid E - T \mid T \\
T & \rightarrow & T * F \mid T \;/\; F \mid F \\
F & \rightarrow & (\; E \;) \mid \mathbf{id}
\end{array}
$$

Derivations

- **Example** $E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid \mathbf{id}$
  - A derivation of -(**id**) from E is: $E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(\mathbf{id})$ این اشتقاق است:
- If $S \overset{*}{\Rightarrow} \alpha$, $^{+}$ where S is the start symbol of a grammar G, we say that α is a **sentential form** of G
  - A sentential form may contain both terminals and nonterminals
  - ** The strings E, -E, -(E), -(id + id) are all sentential forms
- A **sentence** of G is a sentential form with no nonterminals
- **The language generated by a grammar is its set of sentences**
- **Example**

$$E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E + E) \Rightarrow -(\mathbf{id} + E) \Rightarrow -(\mathbf{id} + \mathbf{id})$$

-
مفهوم اشتقاق روی گرامر:

از S می تونیم برسیم به الفا توی یک تعدادی مرحله --> +

به اون عبارتی که از درجه اشتقاق بهش می رسیم اگر به یک عبارتی برسیم که کامل از terminal ها تشکیل شده مثل ** به این گفته میشه sentential

Derivations

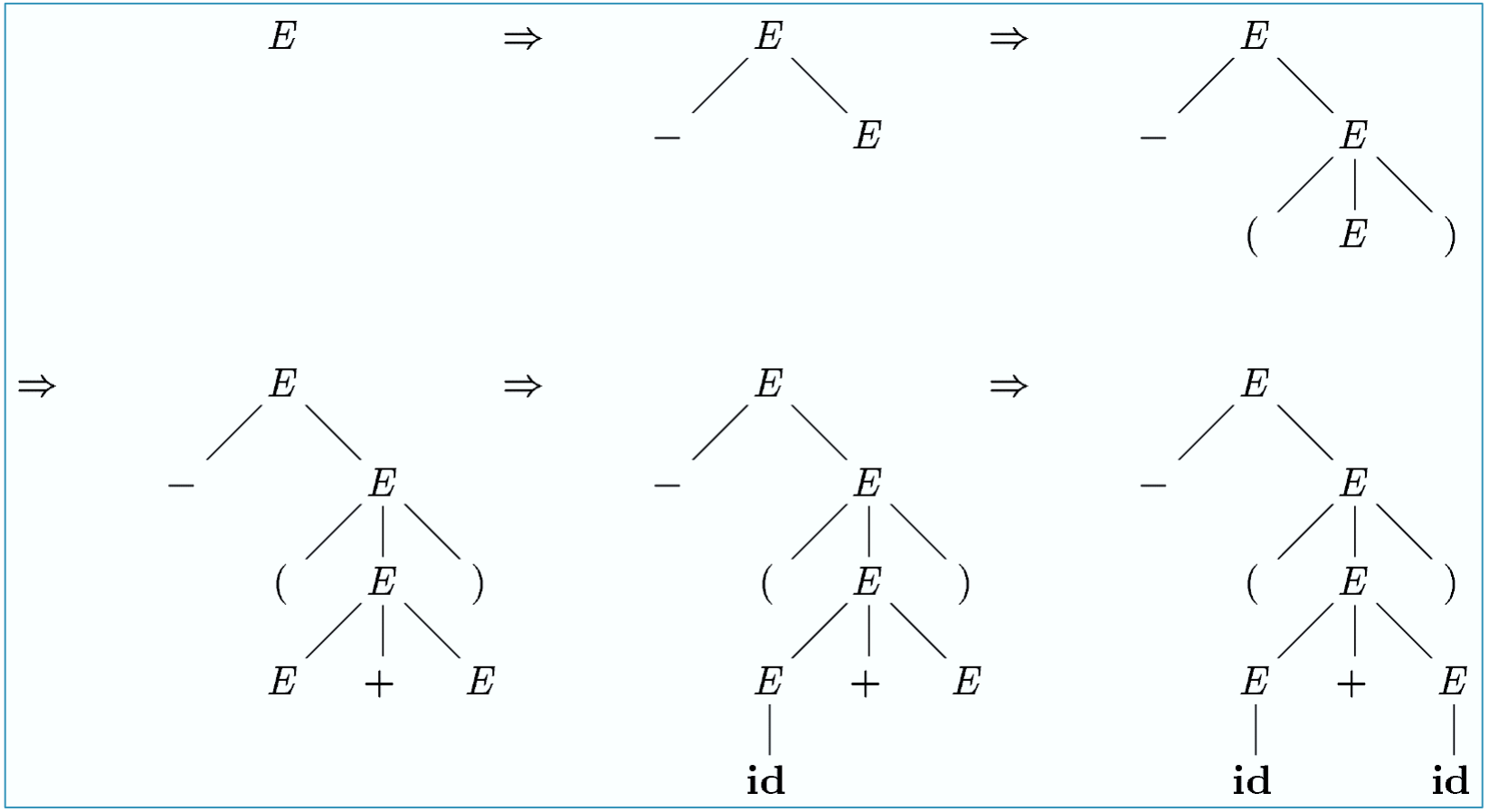
- **Leftmost derivations** $\alpha \underset{lm}{\Rightarrow} \beta$
  - The leftmost nonterminal in each sentential is always chosen
- **Rightmost derivations** $\alpha \underset{rm}{\Rightarrow} \beta$
  - The rightmost nonterminal in each sentential is always chosen
- **Example**

$$E \underset{lm}{\Rightarrow} -E \underset{lm}{\Rightarrow} -(E) \underset{lm}{\Rightarrow} -(E+E) \underset{lm}{\Rightarrow} -(\mathbf{id}+E) \underset{lm}{\Rightarrow} -(\mathbf{id}+\mathbf{id})$$

-
اشتقاق سمت چپ ترین:
سمت چپ ترین nonterminal هر دفعه بیاد جایگزین بشه
اشتقاق سمت راست ترین:
سمت راست ترین nonterminal هر دفعه بیاد جایگزین بشه

Parse Trees

- A parse tree is a graphical representation of a derivation that filters out the order in which productions are applied to replace nonterminals

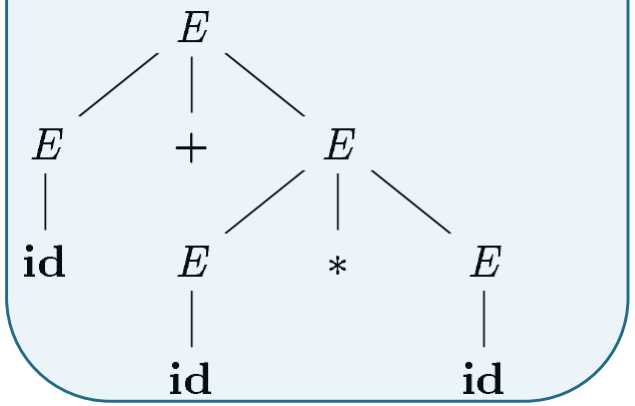


Ambiguity

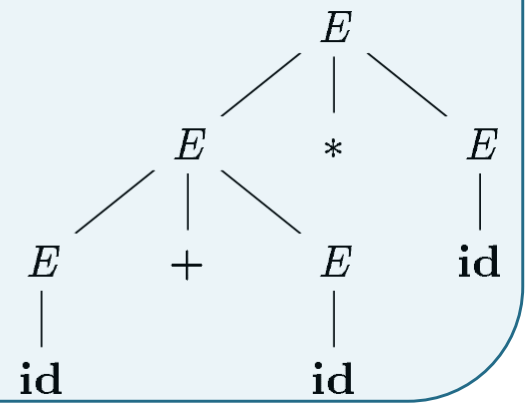
- A grammar that produces more than one parse tree for some sentence is said to be **ambiguous**
  - An ambiguous grammar is one that produces more than one leftmost derivation or more than one rightmost derivation for the same sentence

$$E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid \mathbf{id}$$

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow \mathbf{id} + E \\ &\Rightarrow \mathbf{id} + E * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id} \end{aligned}$$



$$\begin{aligned} E &\Rightarrow E * E \\ &\Rightarrow E + E * E \\ &\Rightarrow \mathbf{id} + E * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * E \\ &\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id} \end{aligned}$$



-

این اشتقاقی که می تونیم برای هر رشته ای داشته باشیم از یک گرامر --> ما اگر حالتی داشته باشیم برای یک گرامر که برای یک رشته یا تعدادی از رشته ها بیشتر از یک اشتقاق بتونیم داشته باشیم اون گرامر میشه گرامر مبهم و اگر فقط یک اشتقاق داشته باشیم اون گرامر مبهم نیست