CS 314 Principles of Programming Languages

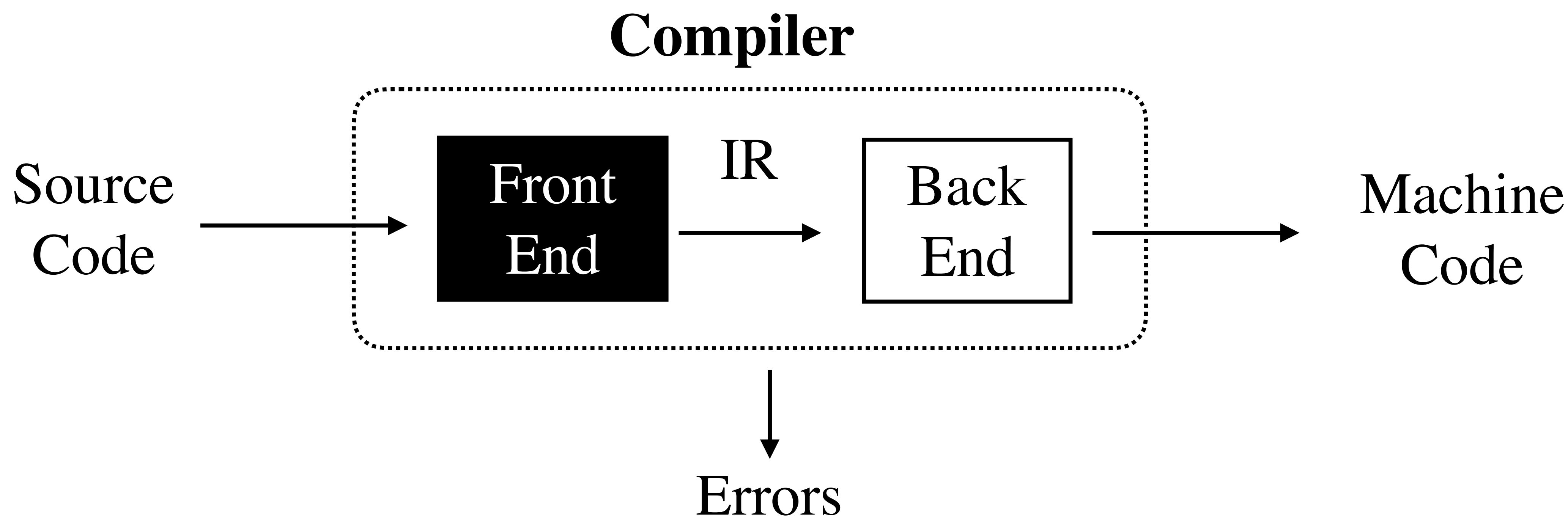
Lecture 4: Syntax Analysis (Parsing)

Prof. Zheng Zhang

Rutgers University

September 14, 2018

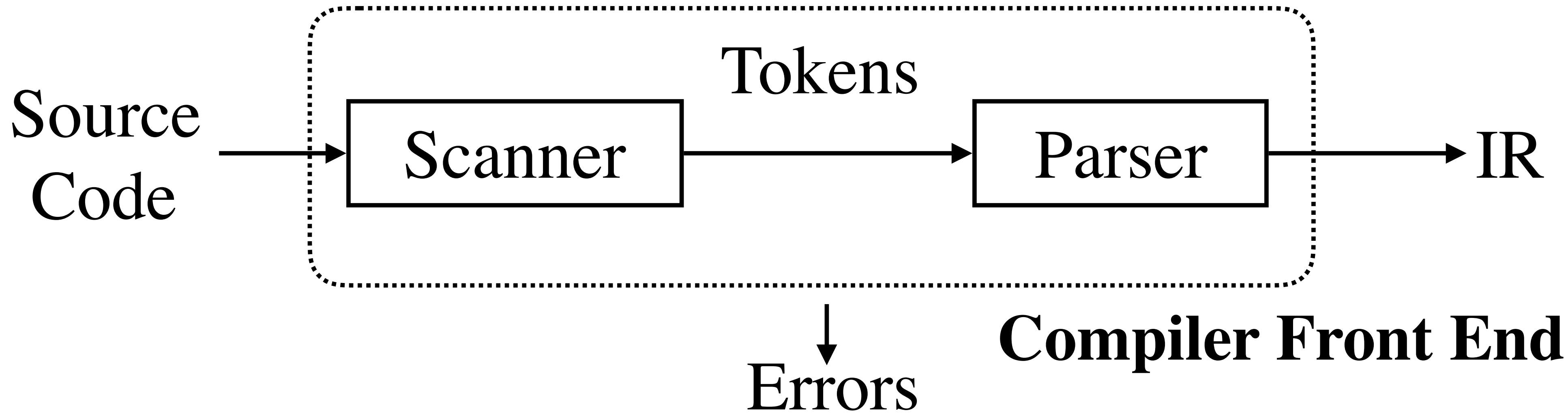
Review: Overview of Compilation



Compiler is in charge of translation:

- Intermediate representation (IR)
- Front end maps legal code into IR
- Back end maps IR onto target machine

Review: Compiler Front End

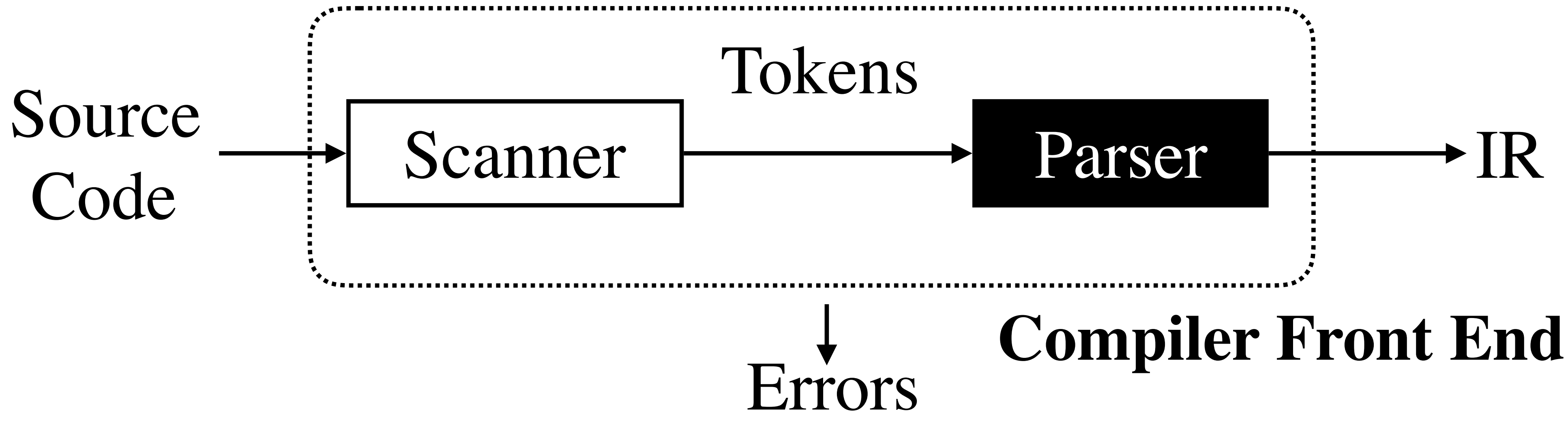


Syntax & semantic analyzer, IR code generator

Front End Responsibilities:

- Recognize legal programs
- Report errors
- Produce intermediate representation

Review: Compiler Front End

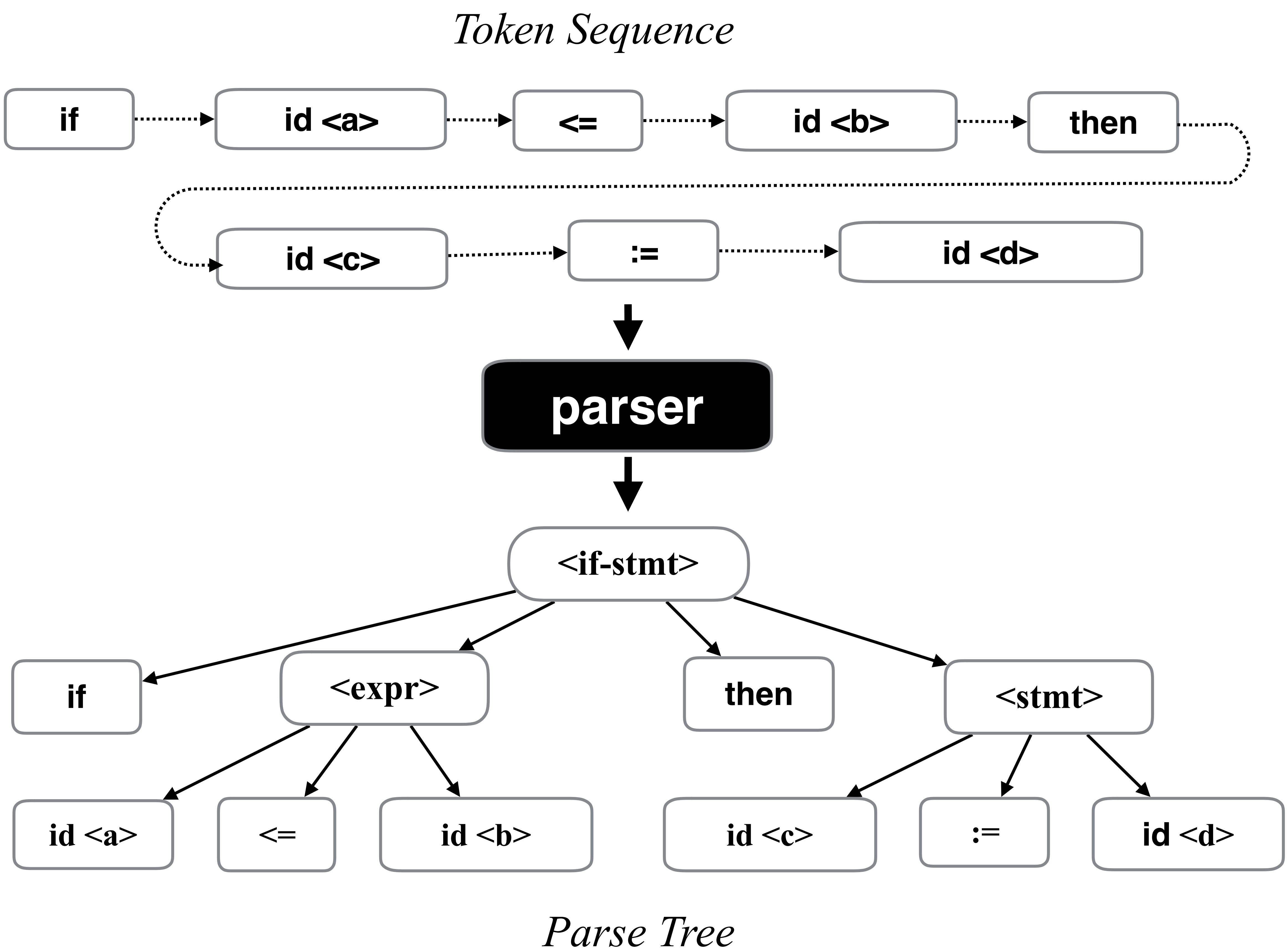


Syntax & semantic analyzer, IR code generator

Front End Responsibilities:

- Recognize legal programs
- Report errors
- Produce intermediate representation

Review: Parsing Example



Review: Context Free Grammars (CFGs)

- A formalism to for describing languages
- A CFG $\mathcal{G} = <\mathbf{T}, \mathbf{N}, \mathbf{P}, \mathbf{S}>$:
  1. A set T of terminal symbols (tokens).
  2. A set N of nonterminal symbols.
  3. A set P production (rewrite) rules.
  4. A special start symbol S.
- The language $L(\mathcal{G})$ is the set of sentences of terminal symbols in T* that can be derived from the start symbol S:

$$L(\mathcal{G}) = \{w \in T^* \mid S \Rightarrow^* w\}$$

Elements of BNF Syntax

Terminal Symbol: **Symbol-in-Boldface**

Non-Terminal Symbol: *Symbol-in-Angle-Brackets*

Production Rule: Non-Terminal ::= Sequence of Symbols

or

Non-Terminal ::= Sequence | Sequence |

…

Example:

…

<if-stmt> ::= **if** <expr> **then** <stmt>

<expr> ::= **id** **<=** **id**

<stmt> ::= **id** **:=** **num**

An Example of a Partial Context Free Grammar

…

<if-stmt> ::= **if** <expr> **then** <stmt>
<expr> ::= **id** **<=** **id**
<stmt> ::= **id** **:=** **num**

…

Context free grammar

Rule 1	**$1 ⇒ 1&**
Rule 2	**$0 ⇒ 0$**
Rule 3	**&1 ⇒ 1$**
Rule 4	**&0 ⇒ 0&**
Rule 5	**$# ⇒ →A**
Rule 6	**&# ⇒ →B**

Not a context free grammar

CFGs are rewrite systems with restrictions on the form of rewrite (production) rules that can be used. The left hand side of a production rule can only be **one non-terminal symbol**.

How a BNF Grammar Describes a Language

A sentence is a sequence of terminal symbols (tokens).

The language L($\mathcal{G}$) of a BNF grammar $\mathcal{G}$ is the set of sentences generated using the grammar:

- Begin with start symbol.
- Iteratively replace non-terminals with terminals according to the rules (rewrite system).

This replacing process is called a derivation ($\Rightarrow$).

Zero or multiple derivation steps are written as $\Rightarrow^*$.

Formally, $L(\mathcal{G}) = \{w \in T^* \mid S \Rightarrow^* w\}$

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), in another word, can X2 := 0 be derived in $\mathcal{G}$?

Start Symbol : <stmt>

↓

⋮ ?

↓

X2 := 0

Grammar $\mathcal{G}$:

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

In which order to apply the rules?

Typically, there are three options:

leftmost ($\Rightarrow_L$), rightmost ($\Rightarrow_R$), any ($\Rightarrow$)

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation	rule
<stmt> $\Rightarrow_L$	
X2 := 0	

1. $<$ letter $>$ ::= A | B | C | … | Z
2. $<$ digit $>$ ::= 0 | 1 | 2 | … | 9
3. $<$ identifier $>$ ::= $<$ letter $>$ |
4. $<$ identifier $><$ letter $>$ |
5. $<$ identifier $><$ digit $>$
6. $<$ stmt $>$ ::= $<$ identifier $>$:= 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation	rule
<stmt> $\Rightarrow_L$	6
<identifier> := 0 $\Rightarrow_L$	
X2 := 0	

1. < letter > ::= A \| B \| C \| … \| Z
2. < digit > ::= 0 \| 1 \| 2 \| … \| 9
3. < identifier > ::= < letter > \|
4. < identifier > < letter > \|
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	
X2 := 0		

1. $<$ letter $>$ ::= A | B | C | … | Z
2. $<$ digit $>$ ::= 0 | 1 | 2 | … | 9
3. $<$ identifier $>$ ::= $<$ letter $>$ |
4. $<$ identifier $><$ letter $>$ |
5. $<$ identifier $><$ digit $>$
6. $<$ stmt $>$ ::= $<$ identifier $>$:= 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation	rule
<stmt> $\Rightarrow_L$	6
<identifier> := 0 $\Rightarrow_L$	5
<identifier> <digit> := 0 $\Rightarrow_L$	3
<letter> <digit> := 0 $\Rightarrow_L$	1
X <digit> := 0 $\Rightarrow_L$	
X2 := 0	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation	rule
<stmt> $\Rightarrow_L$	6
<identifier> := 0 $\Rightarrow_L$	5
<identifier> <digit> := 0 $\Rightarrow_L$	3
<letter> <digit> := 0 $\Rightarrow_L$	1
X <digit> := 0 $\Rightarrow_L$	
X2 := 0	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation	rule
<stmt> $\Rightarrow_L$	6
<identifier> := 0 $\Rightarrow_L$	5
<identifier> <digit> := 0 $\Rightarrow_L$	3
<letter> <digit> := 0 $\Rightarrow_L$	1
X <digit> := 0 $\Rightarrow_L$	2
X2 := 0	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation	rule
<stmt> $\Rightarrow_R$	
X2 := 0	

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation	rule
<stmt> $\Rightarrow_L$	6
<identifier> := 0 $\Rightarrow_L$	5
<identifier> <digit> := 0 $\Rightarrow_L$	3
<letter> <digit> := 0 $\Rightarrow_L$	1
X <digit> := 0 $\Rightarrow_L$	2
X2 := 0	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation	rule
<stmt> $\Rightarrow_R$	6
<identifier> := 0 $\Rightarrow_R$	
X2 := 0	

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation		rule
<stmt>	$\Rightarrow_R$	6
<identifier> := 0	$\Rightarrow_R$	
X2 := 0		

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation		rule
<stmt>	$\Rightarrow_R$	6
<identifier> := 0	$\Rightarrow_R$	5
<identifier> <digit> := 0	$\Rightarrow_R$	
X2 := 0		

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation	rule
<stmt> $\Rightarrow_L$	6
<identifier> := 0 $\Rightarrow_L$	5
<identifier> <digit> := 0 $\Rightarrow_L$	3
<letter> <digit> := 0 $\Rightarrow_L$	1
X <digit> := 0 $\Rightarrow_L$	2
X2 := 0	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation	rule
<stmt> $\Rightarrow_R$	6
<identifier> := 0 $\Rightarrow_R$	5
<identifier><digit> := 0 $\Rightarrow_R$	
X2 := 0	

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation		rule
<stmt>	$\Rightarrow_R$	6
<identifier> := 0	$\Rightarrow_R$	5
<identifier> <digit> := 0	$\Rightarrow_R$	2
<identifier> 2 := 0	$\Rightarrow_R$	

X2 := 0	

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation	rule
<stmt> $\Rightarrow_L$	6
<identifier> := 0 $\Rightarrow_L$	5
<identifier> <digit> := 0 $\Rightarrow_L$	3
<letter> <digit> := 0 $\Rightarrow_L$	1
X <digit> := 0 $\Rightarrow_L$	2
X2 := 0	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation	rule
<stmt> $\Rightarrow_R$	6
<identifier> := 0 $\Rightarrow_R$	5
<identifier> <digit> := 0 $\Rightarrow_R$	2
<identifier> 2 := 0 $\Rightarrow_R$	

X2 := 0	

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 $\in$ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation		rule
<stmt>	$\Rightarrow_R$	6
<identifier> := 0	$\Rightarrow_R$	5
<identifier> <digit> := 0	$\Rightarrow_R$	2
<identifier> 2 := 0	$\Rightarrow_R$	3
<letter> 2 := 0	$\Rightarrow_R$	
X2 := 0		

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation		rule
<stmt>	$\Rightarrow_R$	6
<identifier> := 0	$\Rightarrow_R$	5
<identifier> <digit> := 0	$\Rightarrow_R$	2
<identifier> 2 := 0	$\Rightarrow_R$	3
<letter> 2 := 0	$\Rightarrow_R$	
X2 := 0		

Derivation in a Grammar ($\mathcal{G}$)

Is X2 := 0 ∈ L($\mathcal{G}$), i.e., can X2 := 0 be derived in $\mathcal{G}$?

leftmost derivation		rule
<stmt>	$\Rightarrow_L$	6
<identifier> := 0	$\Rightarrow_L$	5
<identifier> <digit> := 0	$\Rightarrow_L$	3
<letter> <digit> := 0	$\Rightarrow_L$	1
X <digit> := 0	$\Rightarrow_L$	2
X2 := 0		

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

rightmost derivation		rule
<stmt>	$\Rightarrow_R$	6
<identifier> := 0	$\Rightarrow_R$	5
<identifier> <digit> := 0	$\Rightarrow_R$	2
<identifier> 2 := 0	$\Rightarrow_R$	3
<letter> 2 := 0	$\Rightarrow_R$	1
X2 := 0		

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
	1
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language $L(\mathcal{G})$

Can we **recognize** X2 := 0 as being in $L(\mathcal{G})$?

X2 := 0	Rule
<letter> 2 := 0	1
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
	3
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<stmt>	

1. $< \text{letter} > ::= \text{A} \mid \text{B} \mid \text{C} \mid \ldots \mid \text{Z}$
2. $< \text{digit} > ::= 0 \mid 1 \mid 2 \mid \ldots \mid 9$
3. $< \text{identifier} > ::= < \text{letter} > \mid$
4. $\quad < \text{identifier} > < \text{letter} > \mid$
5. $\quad < \text{identifier} > < \text{digit} >$
6. $< \text{stmt} > ::= < \text{identifier} > := 0$

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
	2
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Can we **recognize** X2 := 0 as being in L(𝒢)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
	5
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	6

1. $<$ letter $>$ ::= A | B | C | … | Z
2. $<$ digit $>$ ::= 0 | 1 | 2 | … | 9
3. $<$ identifier $>$ ::= $<$ letter $>$ |
4. $<$ identifier $>$ $<$ letter $>$ |
5. $<$ identifier $>$ $<$ digit $>$
6. $<$ stmt $>$ ::= $<$ identifier $>$:= 0

Parsing of a Language L($\mathcal{G}$)

Can we **recognize** X2 := 0 as being in L($\mathcal{G}$)?

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	6

1. < letter > ::= A | B | C | … | Z
2. < digit > ::= 0 | 1 | 2 | … | 9
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > := 0

We will talk about LL(1) grammars and an example parser for a small language (tinyL) that is implemented using mutually recursive procedures (recursive descent parser) in next class.

Parse Trees (in $\mathcal{G}$)

Each internal (non-leaf) node is a nonterminal;
its children are the RHS of a production for that NT.

The parse tree demonstrates that the grammar generates the terminal string on the frontier.

<stmt>

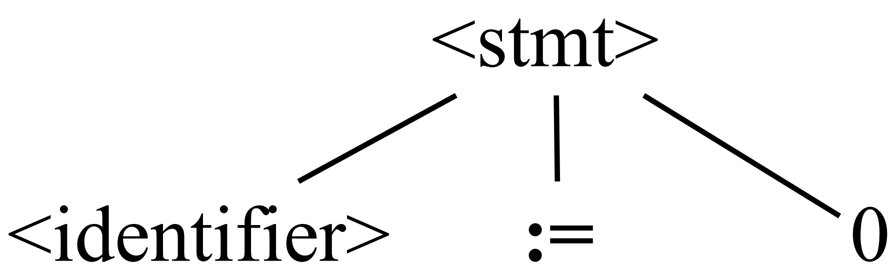
X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	6

A parse tree of X2 := 0 in ($\mathcal{G}$):

Parse Trees (in $\mathcal{G}$)

Each internal (non-leaf) node is a nonterminal;
its children are the RHS of a production for that NT.

The parse tree demonstrates that the grammar generates the terminal string on the frontier.



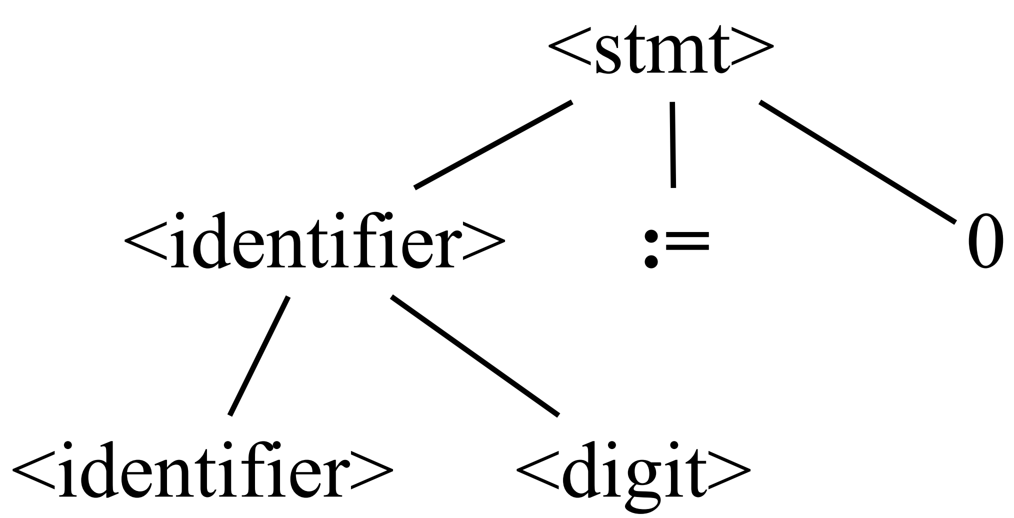
X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	6

A parse tree of X2 := 0 in ($\mathcal{G}$):

Parse Trees (in $\mathcal{G}$)

Each internal (non-leaf) node is a nonterminal;
its children are the RHS of a production for that NT.

The parse tree demonstrates that the grammar generates the terminal string on the frontier.



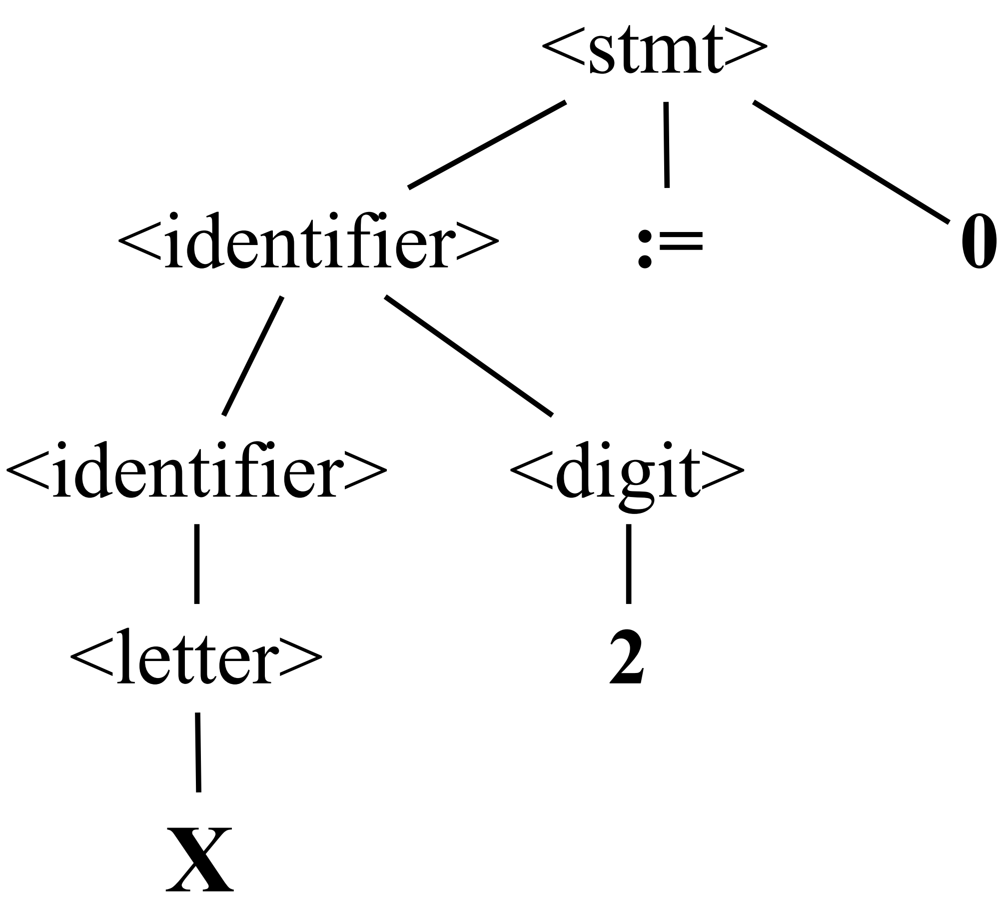
X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	6

A parse tree of X2 := 0 in ($\mathcal{G}$):

Parse Trees (in $\mathcal{G}$)

Each internal node is a nonterminal;
its children are the RHS of a production for that NT.

The parse tree demonstrates that the grammar generates the terminal string on the frontier.



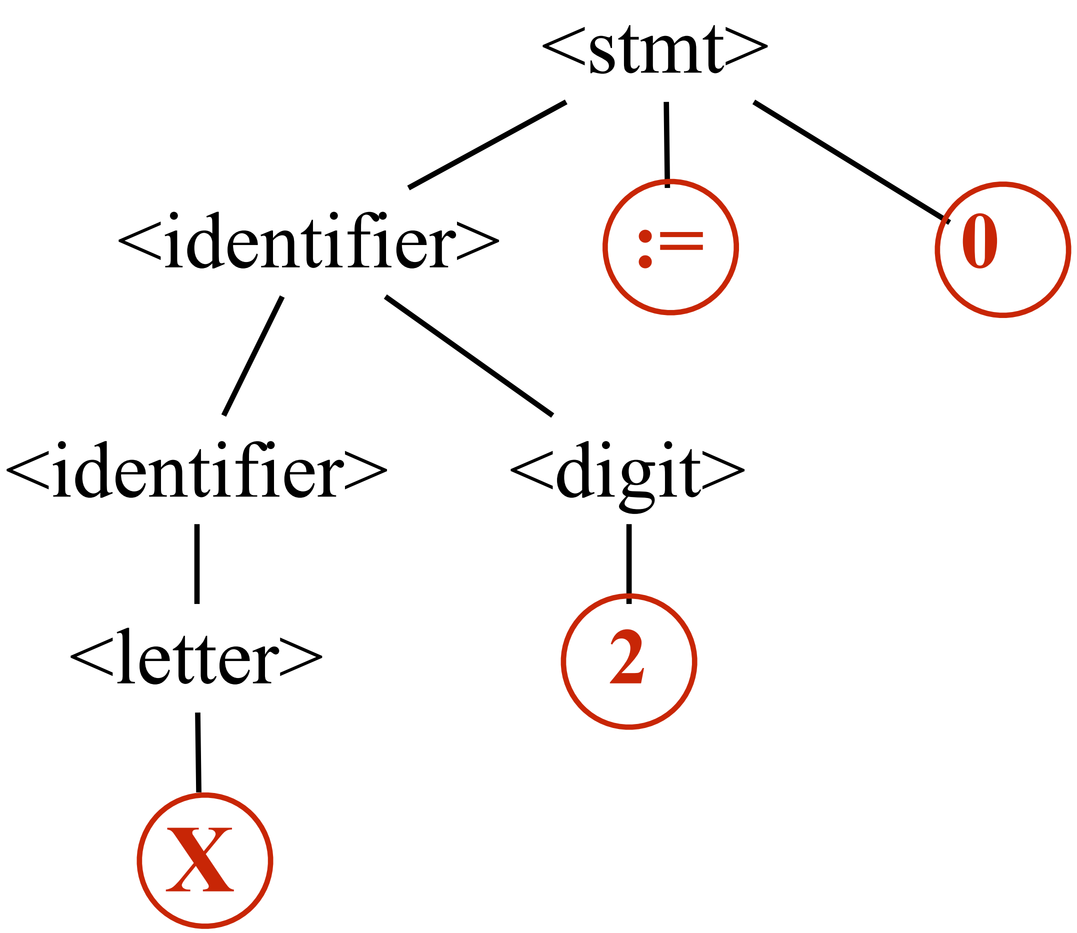
X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	6

A parse tree of X2 := 0 in ($\mathcal{G}$):

Parse Trees (in $\mathcal{G}$)

Each internal node is a nonterminal;
its children are the RHS of a production for that NT.

The parse tree demonstrates that the grammar generates the terminal string on the frontier.



A parse tree of X2 := 0 in ($\mathcal{G}$):

X2 := 0	Rule
<letter> 2 := 0	1
<identifier> 2 := 0	3
<identifier> <digit> := 0	2
<identifier> := 0	5
<stmt>	6

A Language May Have Many Grammars

Consider $\mathcal{G}$’:

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

A Language May Have Many Grammars

Consider $\mathcal{G}$':

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

The Original Grammar $\mathcal{G}$:

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > **:=** 0

How about the parse tree of X2 := 0 in $\mathcal{G}$' and $\mathcal{G}$?

A Language May Have Many Grammars

Consider $\mathcal{G}$':	The Original Grammar $\mathcal{G}$:
1. < letter > ::= **A** | **B** | **C** | … | **Z**	1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**	2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |	3. < identifier > ::= < letter > |
4. < ident > < letterordigit >	4. < identifier > < letter > |
5. < stmt > ::= < ident > **:=** 0	5. < identifier > < digit >
6. < letterordigit > ::= < letter > | < digit >	6. < stmt > ::= < identifier > **:=** 0

<stmt>

X2 := 0

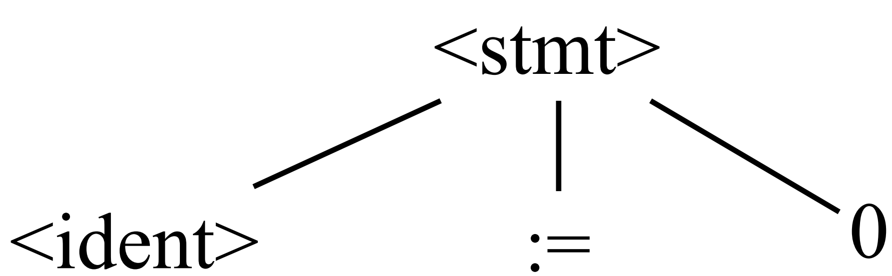
A Language May Have Many Grammars

Consider $\mathcal{G}$':

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

The Original Grammar $\mathcal{G}$:

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > **:=** 0



X2 := 0

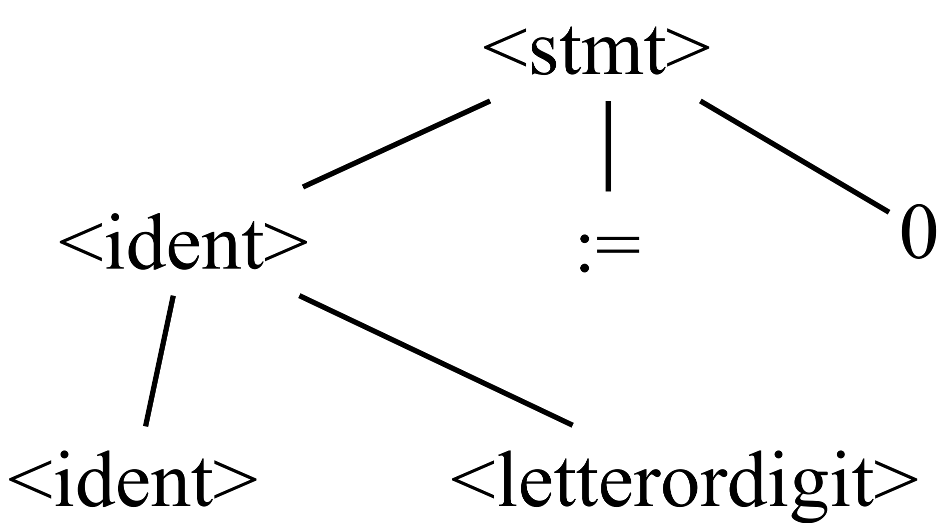
A Language May Have Many Grammars

Consider $\mathcal{G}'$:

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

The Original Grammar $\mathcal{G}$:

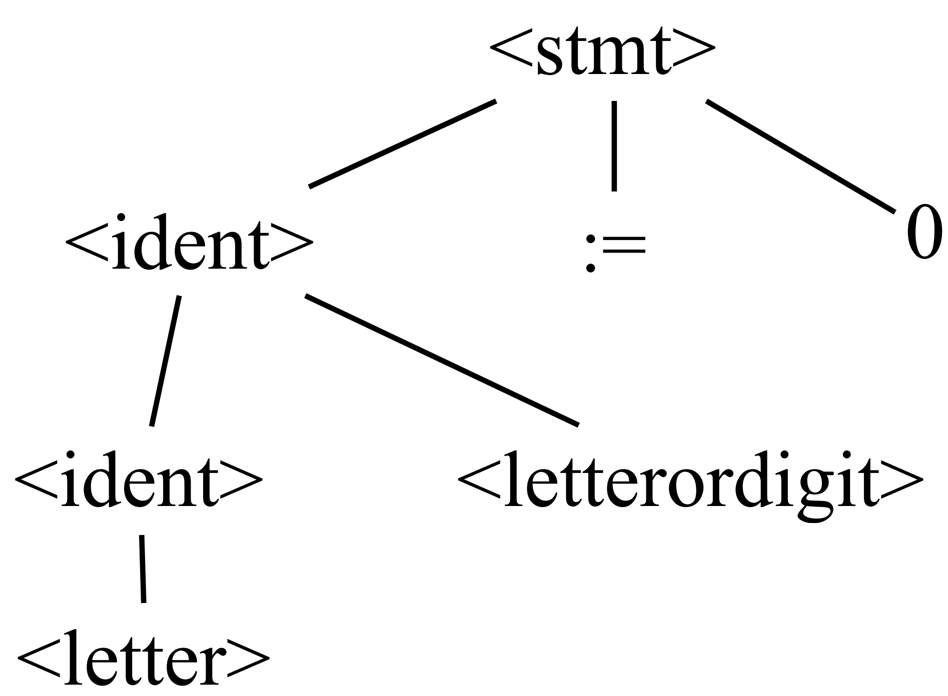
1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > **:=** 0



X2 := 0

A Language May Have Many Grammars

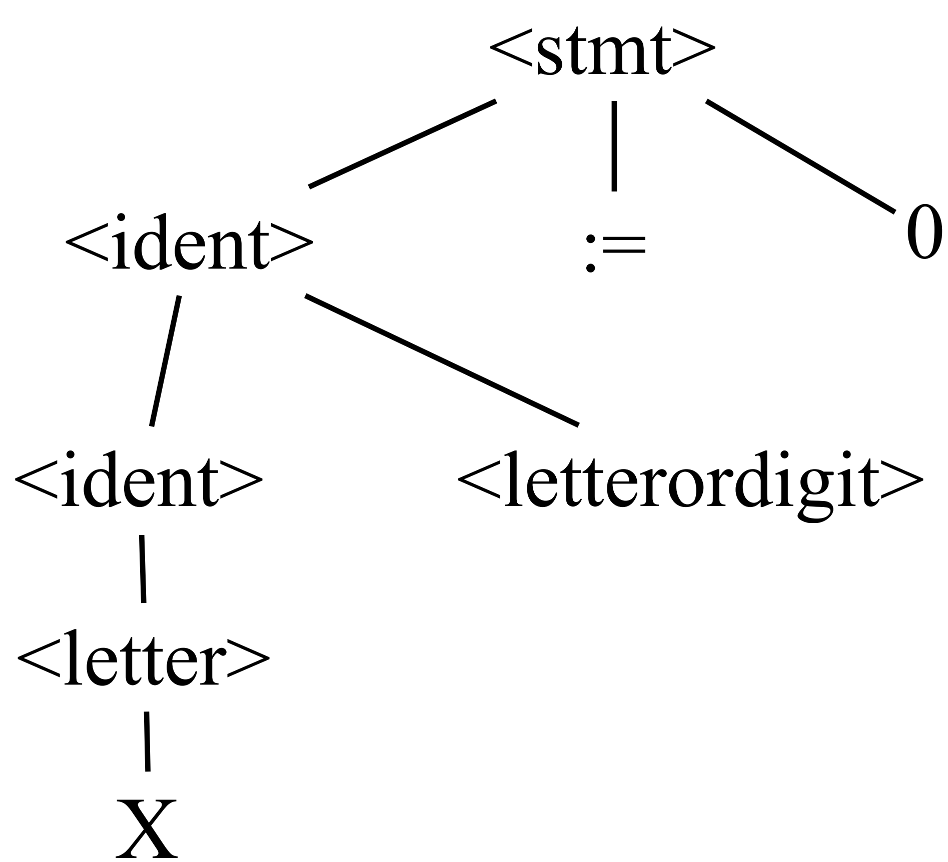
Consider $\mathcal{G}'$:	The Original Grammar $\mathcal{G}$:
1. < letter > ::= **A** | **B** | **C** | … | **Z**	1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**	2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |	3. < identifier > ::= < letter > |
4. < ident > < letterordigit >	4. < identifier > < letter > |
5. < stmt > ::= < ident > **:=** 0	5. < identifier > < digit >
6. < letterordigit > ::= < letter > | < digit >	6. < stmt > ::= < identifier > **:=** 0



X2 := 0

A Language May Have Many Grammars

Consider $\mathcal{G}$’:	The Original Grammar $\mathcal{G}$:
1. < letter > ::= **A** | **B** | **C** | … | **Z**	1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**	2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |	3. < identifier > ::= < letter > |
4. < ident > < letterordigit >	4. < identifier > < letter > |
5. < stmt > ::= < ident > **:=** 0	5. < identifier > < digit >
6. < letterordigit > ::= < letter > | < digit >	6. < stmt > ::= < identifier > **:=** 0



X2 := 0

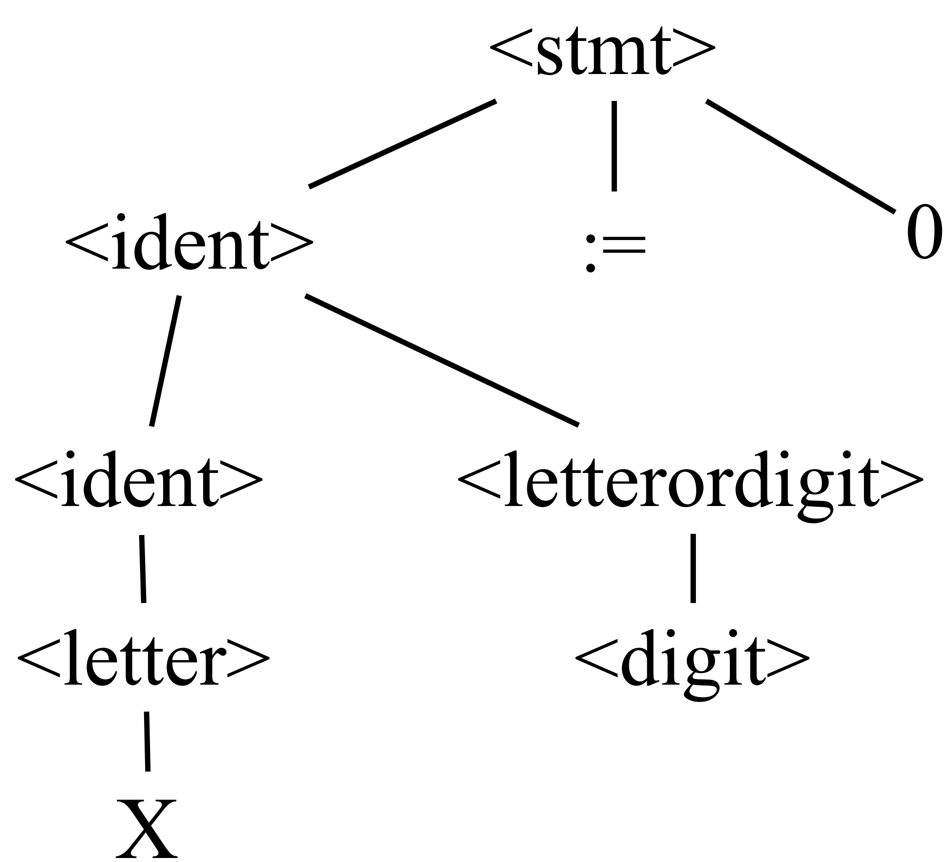
A Language May Have Many Grammars

Consider $\mathcal{G}$':

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

The Original Grammar $\mathcal{G}$:

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > **:=** 0



X2 := 0

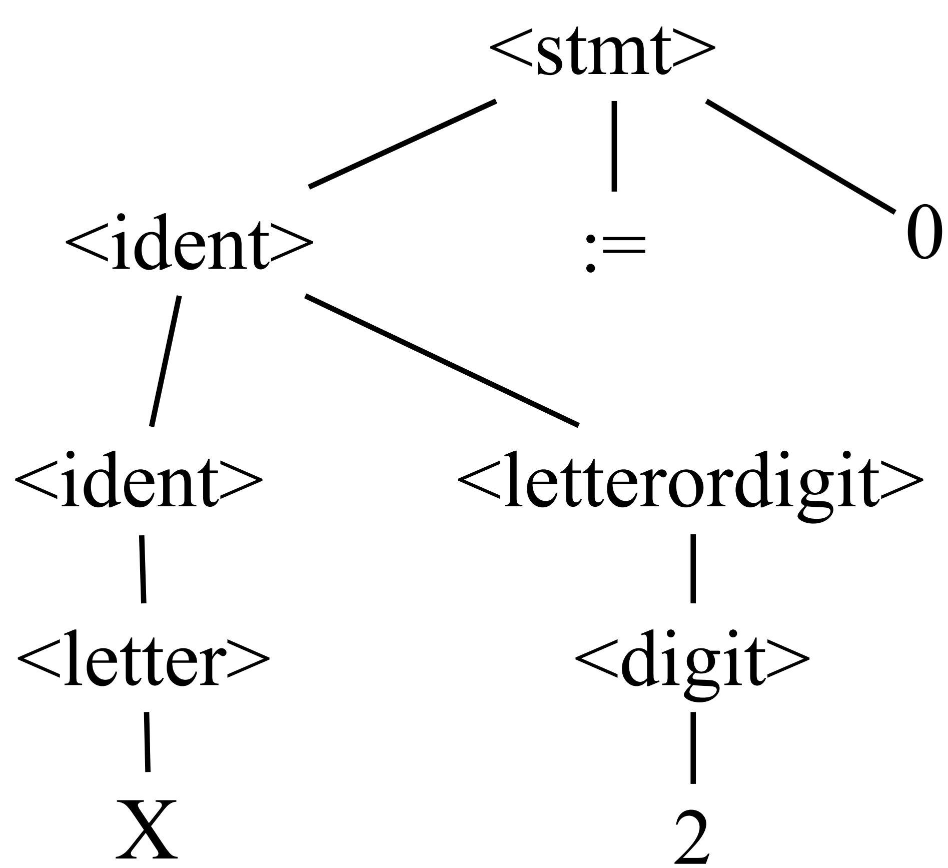
A Language May Have Many Grammars

Consider $\mathcal{G}$':

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

The Original Grammar $\mathcal{G}$:

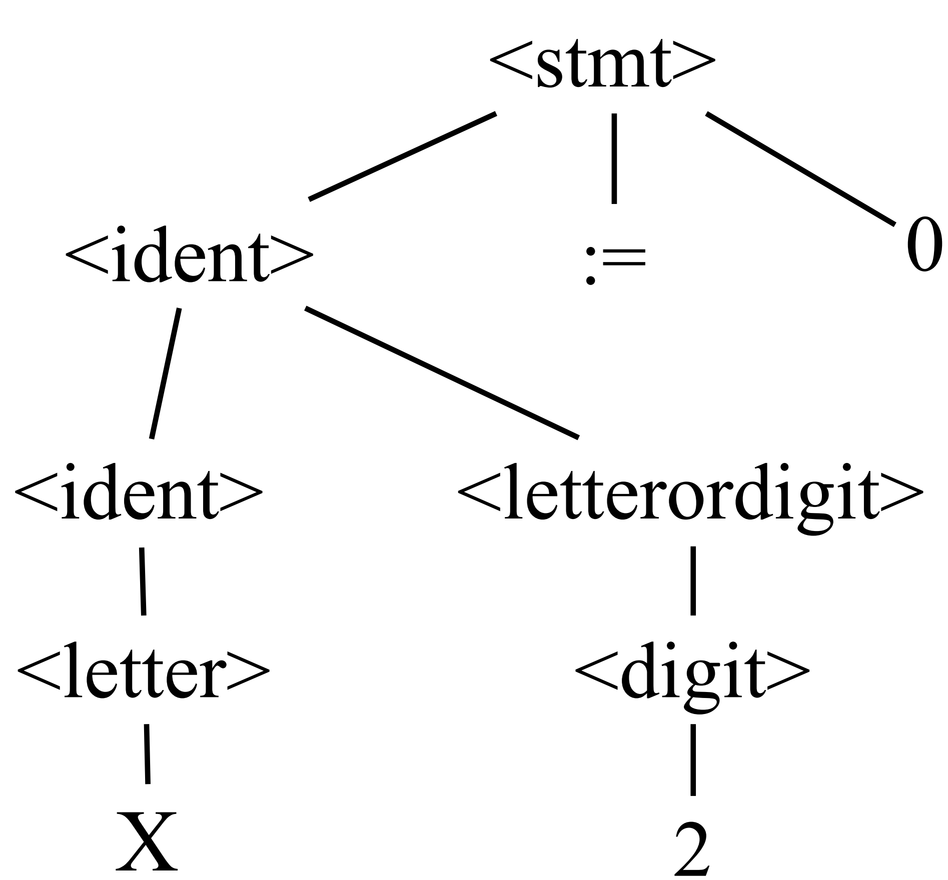
1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > **:=** 0



X2 := 0

A Language May Have Many Grammars

Consider $\mathcal{G}$':	The Original Grammar $\mathcal{G}$:
1. < letter > ::= **A** | **B** | **C** | … | **Z**	1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**	2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |	3. < identifier > ::= < letter > |
4. < ident > < letterordigit >	4. < identifier > < letter > |
5. < stmt > ::= < ident > **:=** 0	5. < identifier > < digit >
6. < letterordigit > ::= < letter > | < digit >	6. < stmt > ::= < identifier > **:=** 0



X2 := 0

<stmt>

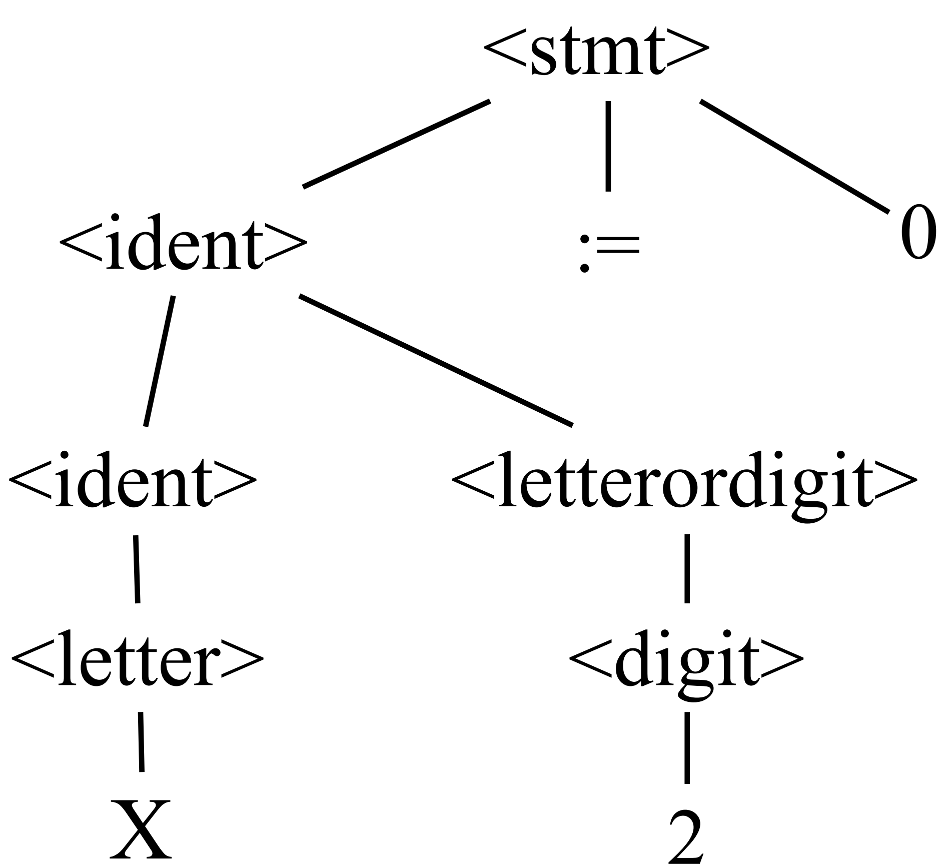
A Language May Have Many Grammars

Consider $\mathcal{G}$':

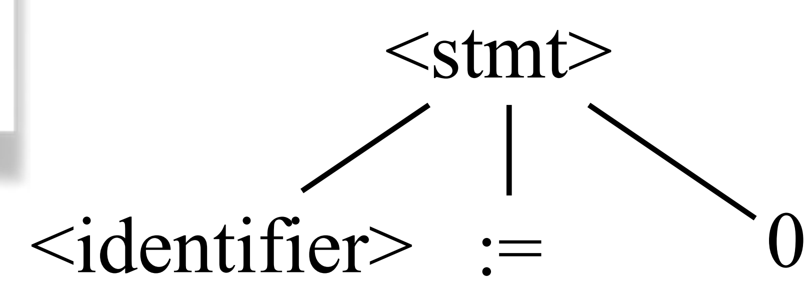
1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

The Original Grammar $\mathcal{G}$:

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > **:=** 0



X2 := 0



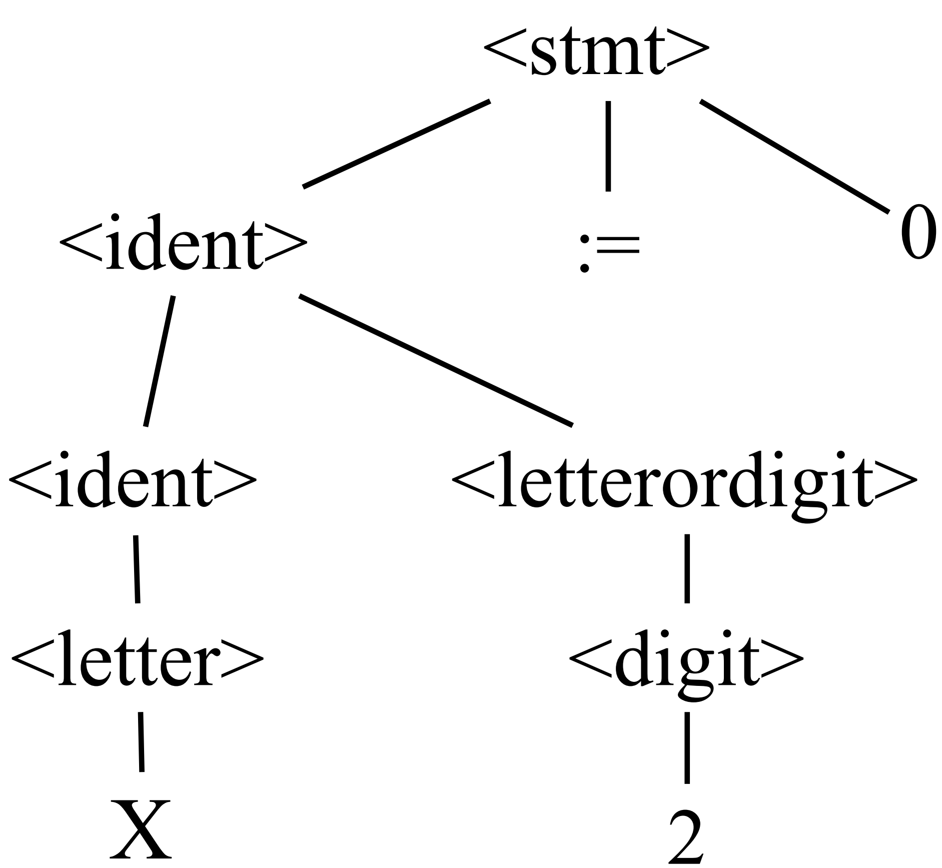
A Language May Have Many Grammars

Consider $\mathcal{G}$':

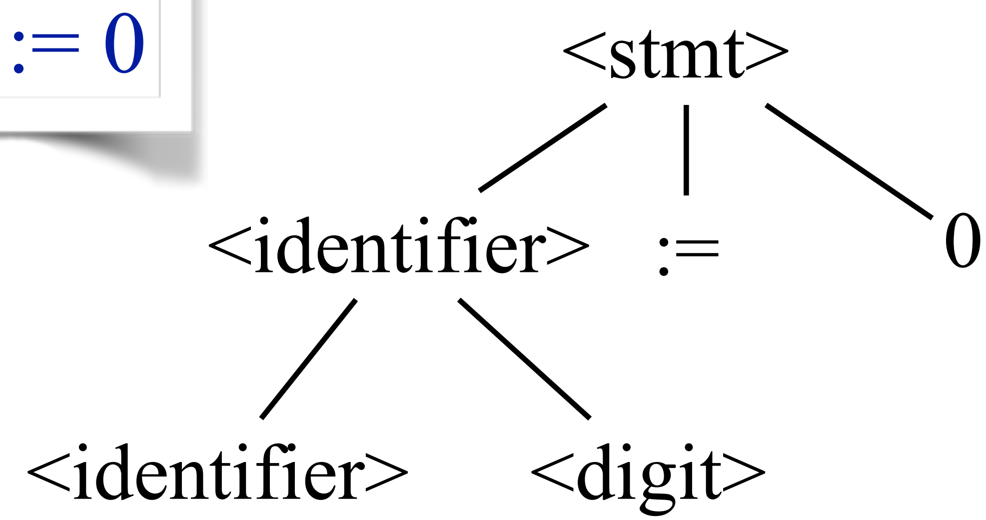
1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

The Original Grammar $\mathcal{G}$:

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > **:=** 0



X2 := 0



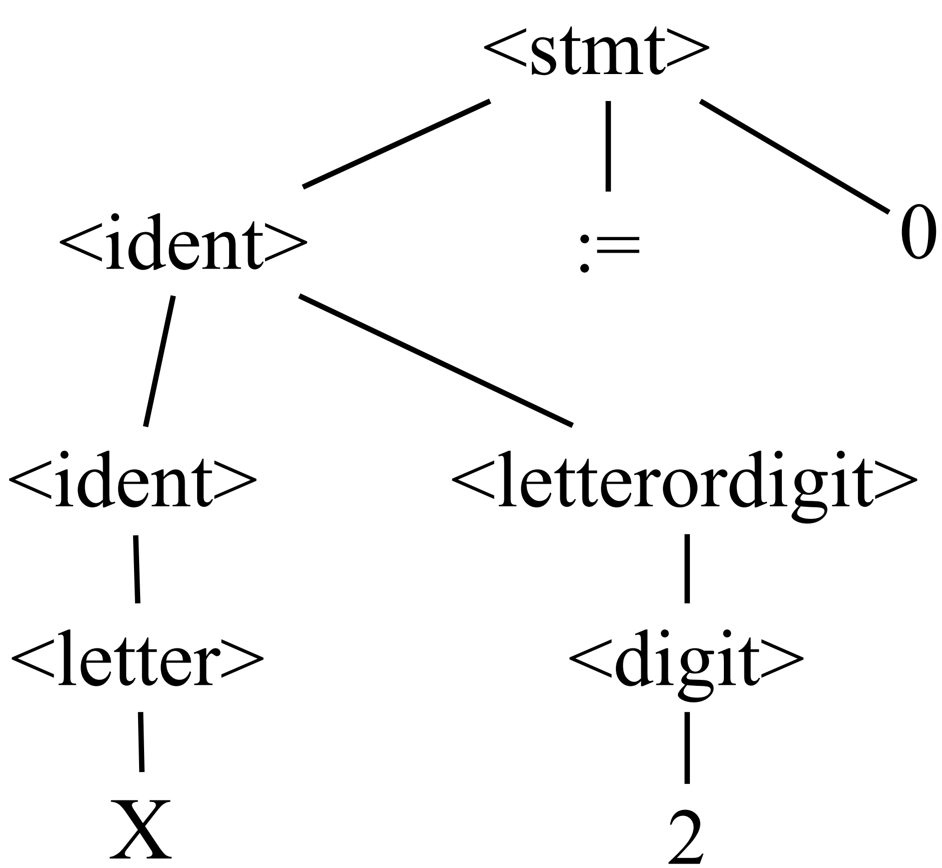
A Language May Have Many Grammars

Consider $\mathcal{G}$':

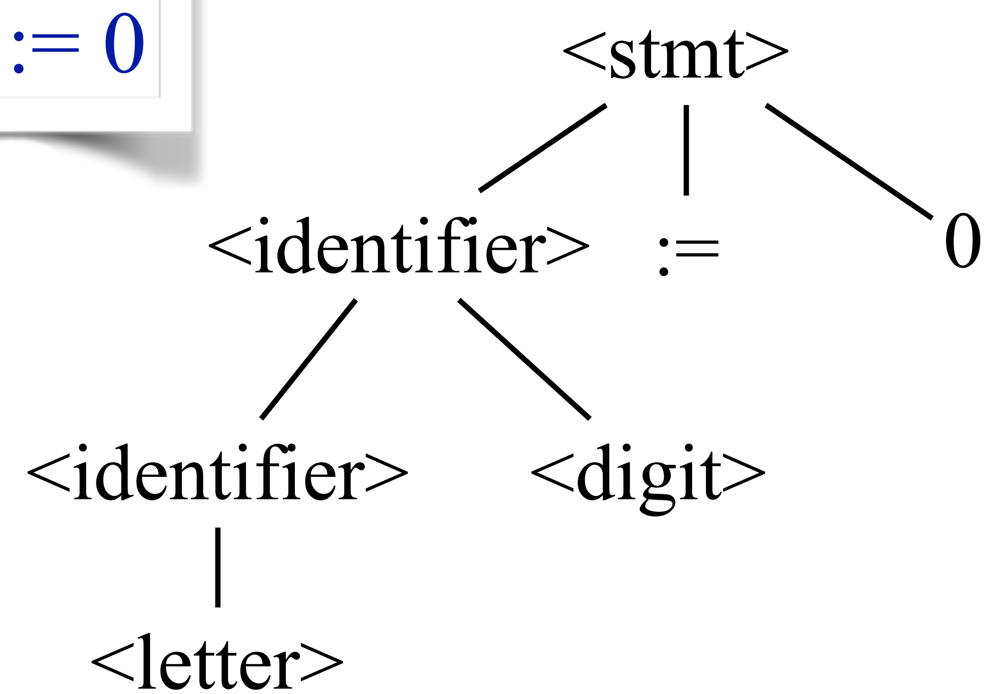
1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |
4. < ident > < letterordigit >
5. < stmt > ::= < ident > **:=** 0
6. < letterordigit > ::= < letter > | < digit >

The Original Grammar $\mathcal{G}$:

1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < identifier > ::= < letter > |
4. < identifier > < letter > |
5. < identifier > < digit >
6. < stmt > ::= < identifier > **:=** 0

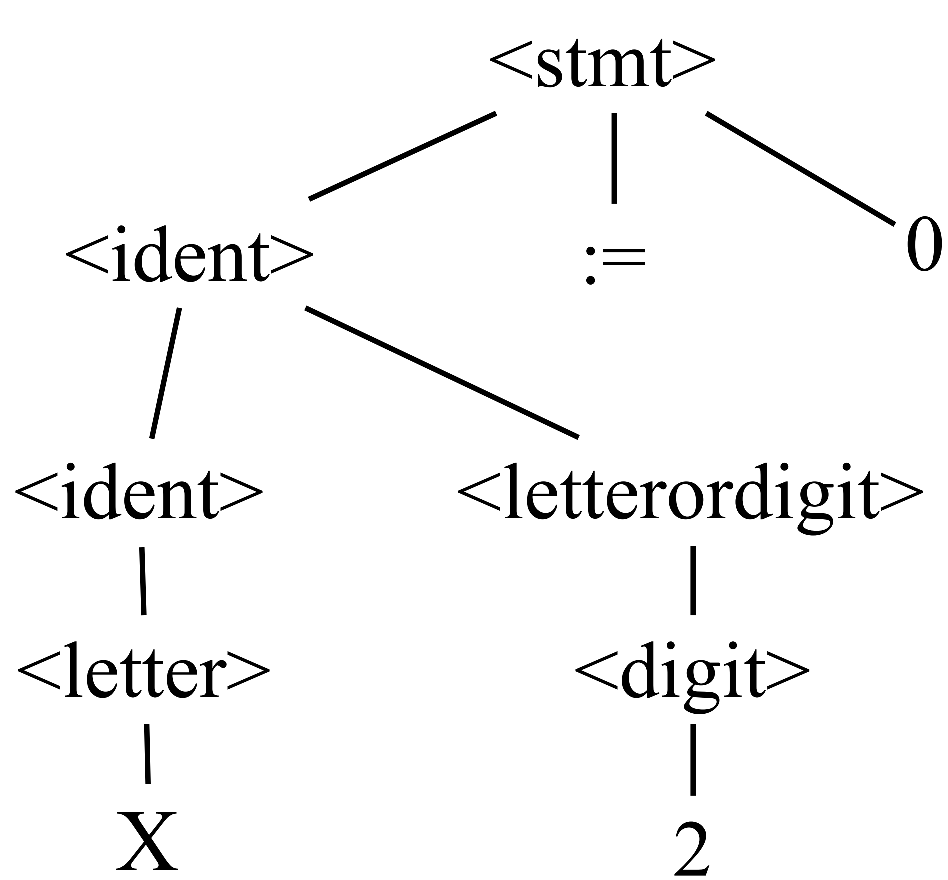


X2 := 0

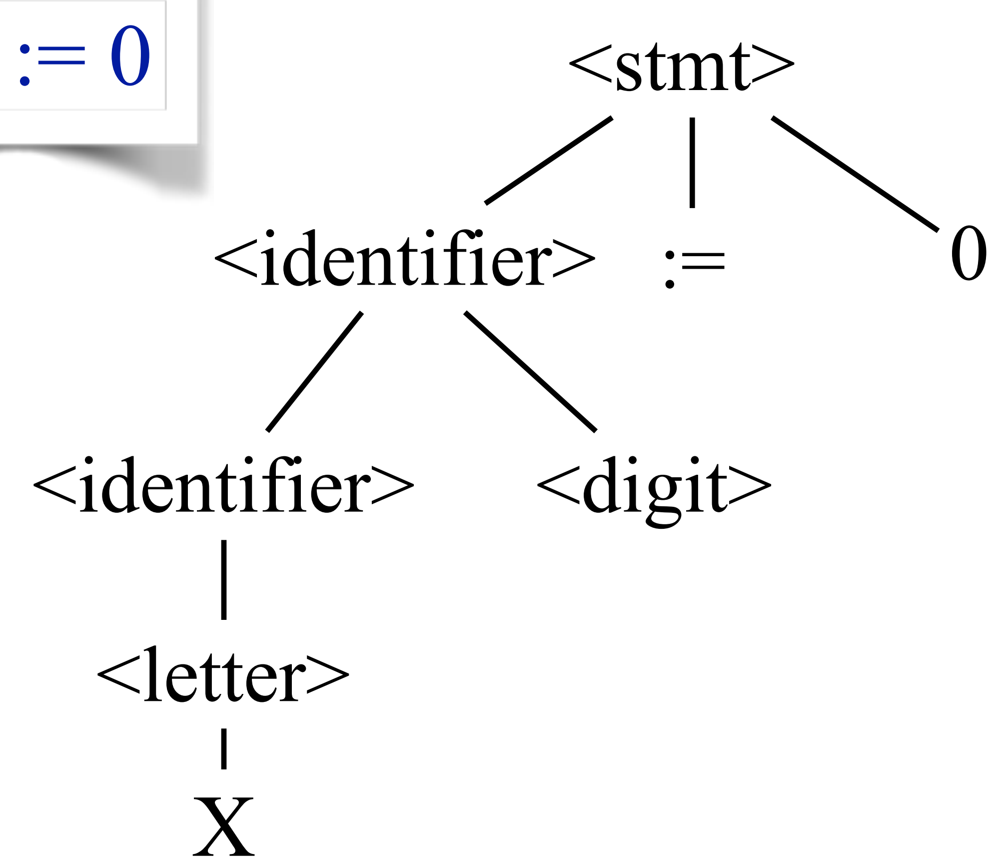


A Language May Have Many Grammars

Consider $\mathcal{G}$':	The Original Grammar $\mathcal{G}$:
1. < letter > ::= **A** | **B** | **C** | … | **Z**	1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**	2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |	3. < identifier > ::= < letter > |
4. < ident > < letterordigit >	4. < identifier > < letter > |
5. < stmt > ::= < ident > **:=** 0	5. < identifier > < digit >
6. < letterordigit > ::= < letter > | < digit >	6. < stmt > ::= < identifier > **:=** 0

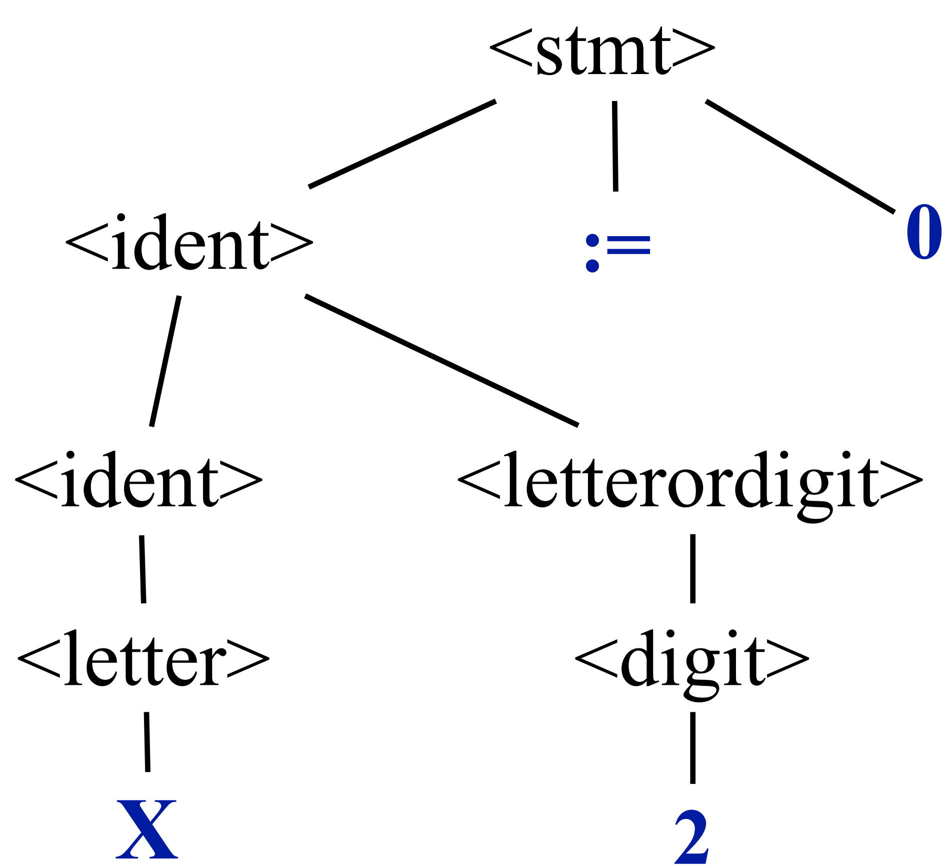


X2 := 0

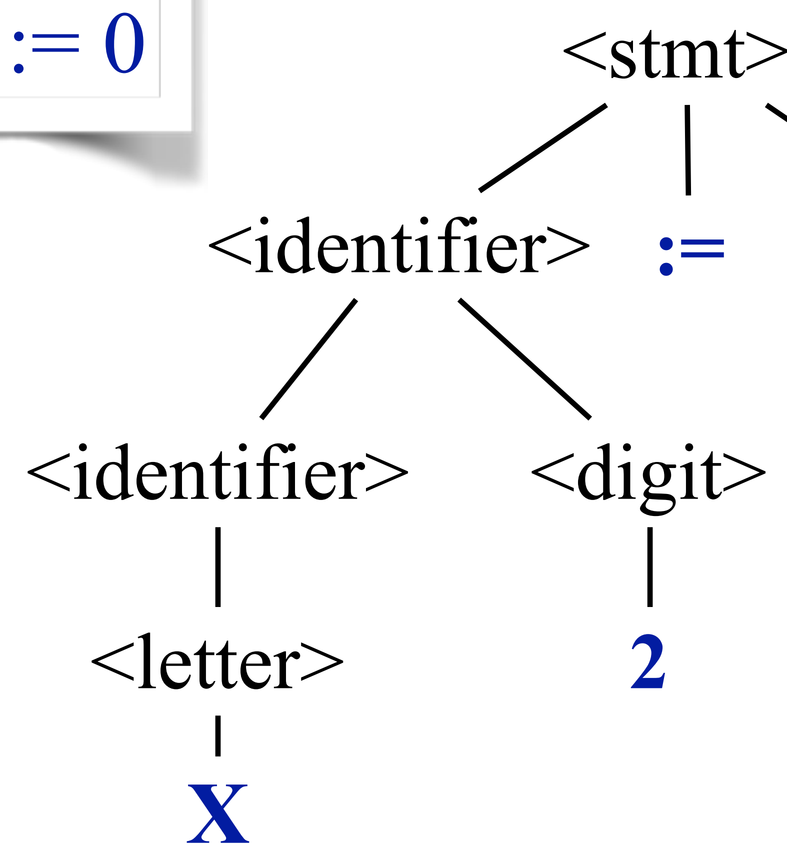


A Language May Have Many Grammars

Consider $\mathcal{G}'$:	The Original Grammar $\mathcal{G}$:
1. < letter > ::= **A** | **B** | **C** | … | **Z**	1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**	2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |	3. < identifier > ::= < letter > |
4. < ident > < letterordigit >	4. < identifier > < letter > |
5. < stmt > ::= < ident > **:=** 0	5. < identifier > < digit >
6. < letterordigit > ::= < letter > | < digit >	6. < stmt > ::= < identifier > **:=** 0



X2 := 0



A Language May Have Many Grammars

Consider $\mathcal{G}$':	The Original Grammar $\mathcal{G}$:
1. < letter > ::= **A** | **B** | **C** | … | **Z**	1. < letter > ::= **A** | **B** | **C** | … | **Z**
2. < digit > ::= **0** | **1** | **2** | … | **9**	2. < digit > ::= **0** | **1** | **2** | … | **9**
3. < ident > ::= < letter > |	3. < identifier > ::= < letter > |
4. < identifier > < letterordigit >	4. < identifier > < letter > |
5. < stmt > ::= < ident > **:=** 0	5. < identifier > < digit >
6. < letterordigit > ::= < letter > | < digit >	6. < stmt > ::= < identifier > **:=** 0

How about the parse tree of X2 := 0 in $\mathcal{G}$ and $\mathcal{G}$?

$\mathcal{G}$ and $\mathcal{G}$' generate the same language, but yield different parse trees.

Grammars and Programming Languages

Many grammars may correspond to one programming language.

Good grammars:

- Captures the logic structure of the language

  ⇒ structure carries some semantic information
  (example: expression grammar)

- Use meaningful names
- Are easy to read
- Are unambiguous
- …

What's problem with ambiguity?

Ambiguous Grammars

*"Time **flies like** an arrow; fruit **flies like** a banana."*

A grammar $\mathcal{G}$ is ambiguous iff there exists a w $\in$ L($\mathcal{G}$) such that there are:

- two distinct parse trees for w, or
- two distinct leftmost derivations for w, or
- two distinct rightmost derivations for w.

We want a unique semantics of our programs, which typically requires a unique syntactic structure.

Simple Statement Grammar

*How are nested **if** statements parsed with this grammar?*

if x == 0 **then if** y == 0 **then** z := 1 **else** z := 2

<start> ::= <stmt>

<stmt> ::= <if-stmt> | <assgn>

<if-stmt> ::= **if** <expr> **then** <stmt> |
if <expr> **then** <stmt> **else** <stmt>

<assgn> ::= <id> **:=** <d>

<expr> ::= <id> **==** 0

<d> ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

<id> ::= **a** | **b** | **c** | … | **z** |

Simple Statement Grammar

*How are nested **if** statements parsed with this grammar?*

if x == 0 **then**[**if** y == 0 **then** z := 1 **else** z := 2]

<start> ::= <stmt>

<stmt> ::= <if-stmt> | <assgn>

<if-stmt> ::= **if** <expr> **then** <stmt> |
if <expr> **then** <stmt> **else** <stmt>

<assgn> ::= <id> **:=** <d>

<expr> ::= <id> **==** 0

<d> ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

<id> ::= **a** | **b** | **c** | … | **z** |

Simple Statement Grammar

*How are nested **if** statements parsed with this grammar?*

if x == 0 **then**[**if** y == 0 **then** z := 1] **else** z := 2

<start> ::= <stmt>

<stmt> ::= <if-stmt> | <assgn>

<if-stmt> ::= **if** <expr> **then** <stmt> |
if <expr> **then** <stmt> **else** <stmt>

<assgn> ::= <id> **:=** <d>

<expr> ::= <id> == 0

<d> ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

<id> ::= **a** | **b** | **c** | … | **z** |

Dangling Else Ambiguity

if x = 0 **then if** y = 0 **then** z := 1 **else** z := 2

How to deal with ambiguity?

Dangling Else Ambiguity

if x = 0 **then if** y = 0 **then** z := 1 **else** z := 2

How to deal with ambiguity?

- Change the language to include delimiters
  Example: Adding new terminal symbols.

  It can fix the *dangling else*, *expression* grammars.

Changing the Language to Include Delimiters

Algol 68 if statement:

<if-stmt> ::= **if** <expr> **then** <stmt> **fi** |

if <expr> **then** <stmt> **else** <stmt> **fi**

How would you use this syntax to express the meaning of the two different parse trees for:

if x = 0 **then** **if** y = 0 **then** z := 1 **else** z := 2

This: **if** x = 0 **then** **if** y = 0 **then** z := 1 **fi** **else** z := 2 **fi**

Or that: **if** x = 0 **then** **if** y = 0 **then** z := 1 **else** z := 2 **fi** **fi**

Dangling Else Ambiguity

if x = 0 **then if** y = 0 **then** z := 1 **else** z := 2

How to deal with ambiguity?

- Change the language to include delimiters
  Example: Adding new terminal symbols.

  It can fix the *dangling else*, *expression* grammars.

Dangling Else Ambiguity

if x = 0 **then if** y = 0 **then** z := 1 **else** z := 2

How to deal with ambiguity?

- Change the language to include delimiters
  Example: Adding new terminal symbols.
  It can fix the *dangling else*, *expression* grammars.
- Change the grammar
  Example: Impose **associativity** and **precedence** in an *arithmetic expression* grammar.

Arithmetic Expression Grammar

Parse “8 - 3 * 2”:

$$
\begin{aligned}
\langle \text{start} \rangle &::= \langle \text{expr} \rangle \\
\langle \text{expr} \rangle &::= \langle \text{expr} \rangle + \langle \text{expr} \rangle \mid \\
&\qquad \langle \text{expr} \rangle - \langle \text{expr} \rangle \mid \\
&\qquad \langle \text{expr} \rangle * \langle \text{expr} \rangle \mid \\
&\qquad \langle \text{expr} \rangle / \langle \text{expr} \rangle \mid \\
&\qquad \langle \text{expr} \rangle \wedge \langle \text{expr} \rangle \mid \\
&\qquad \langle \text{d} \rangle \mid \langle \text{l} \rangle \\
\langle \text{d} \rangle &::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \mathbf{9} \\
\langle \text{l} \rangle &::= \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots \mid \mathbf{z}
\end{aligned}
$$

Possible Parse Trees

Parse "8 - 3 * 2":

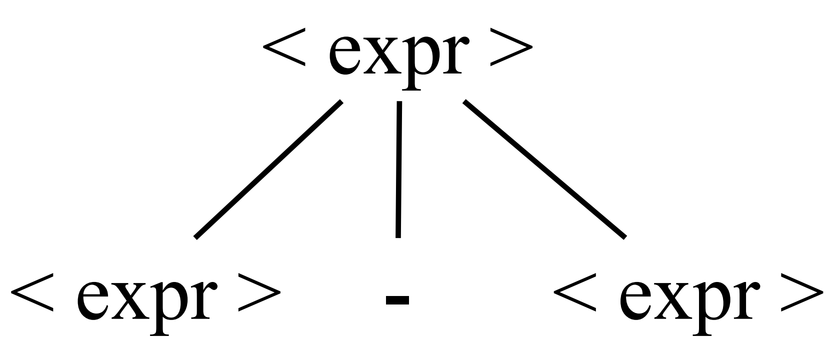
$< \text{expr} >$

$$
\begin{aligned}
< \text{start} > &::= \; < \text{expr} > \\
< \text{expr} > &::= \; < \text{expr} > \mathbf{+} < \text{expr} > \mid \\
&\quad\; < \text{expr} > \mathbf{-} < \text{expr} > \mid \\
&\quad\; < \text{expr} > \mathbf{*} < \text{expr} > \mid \\
&\quad\; < \text{expr} > \mathbf{/} < \text{expr} > \mid \\
&\quad\; < \text{expr} > \mathbf{\wedge} < \text{expr} > \mid \\
&\quad\; < \text{d} > \mid < \text{l} > \\
< \text{d} > &::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \mathbf{9} \\
< \text{l} > &::= \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots \mid \mathbf{z}
\end{aligned}
$$

Parse Tree I

Possible Parse Trees

Parse “8 - 3 * 2”:

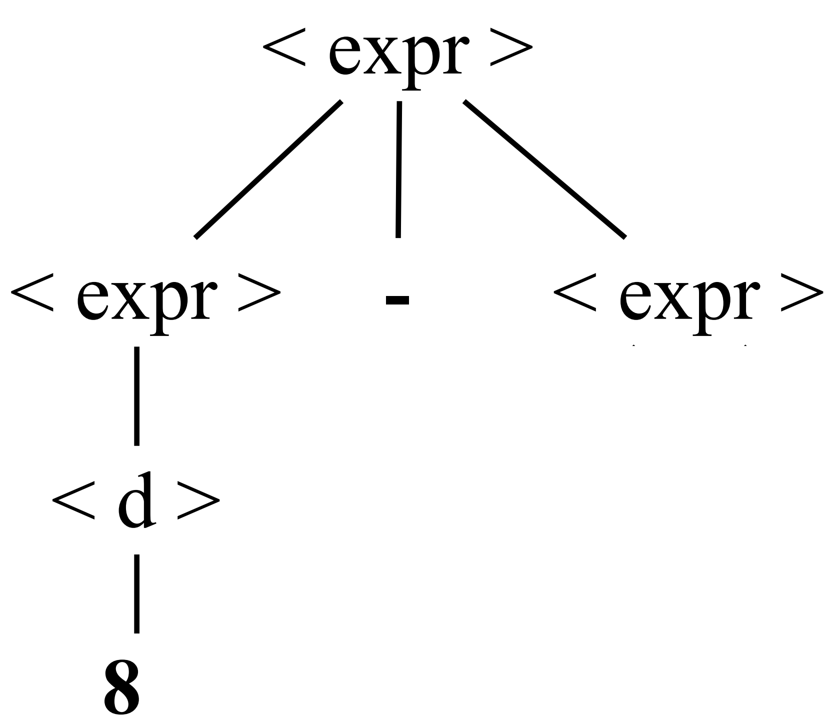


< start > ::= < expr >
< expr > :: = < expr > + < expr > |
< expr > **-** < expr > |
< expr > * < expr > |
< expr > / < expr > |
< expr > ^ < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree I

Possible Parse Trees

Parse “8 - 3 * 2”:

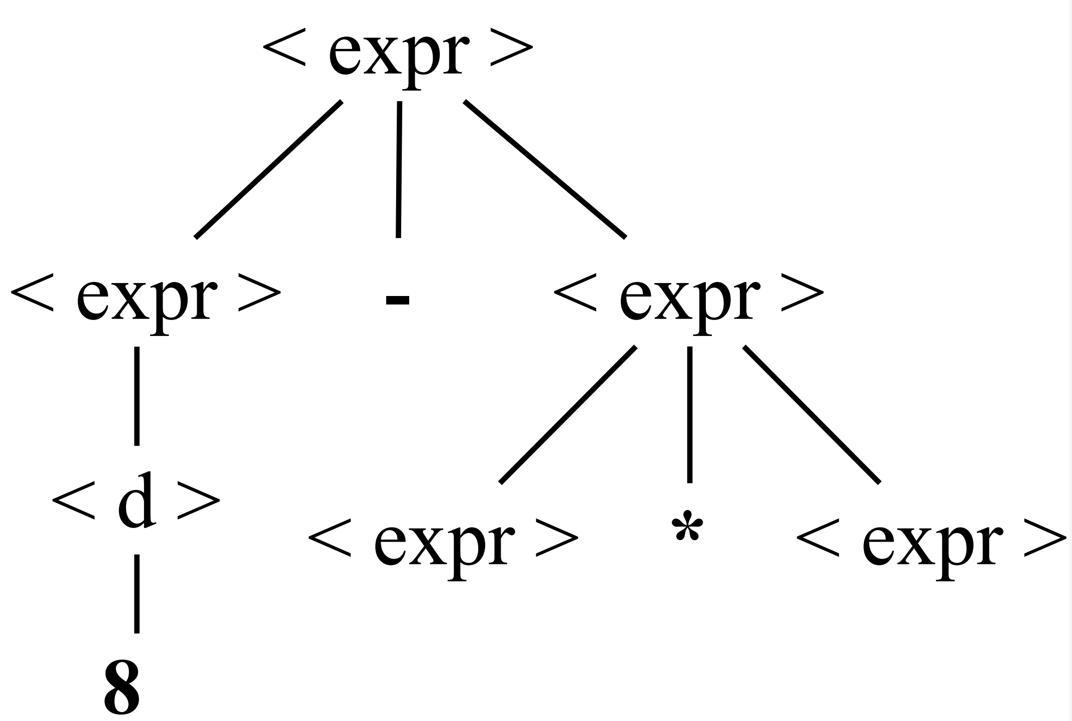


< start > ::= < expr >
< expr > :: = < expr > + < expr > |
< expr > **-** < expr > |
< expr > * < expr > |
< expr > / < expr > |
< expr > ^ < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree I

Possible Parse Trees

Parse "8 - 3 * 2":

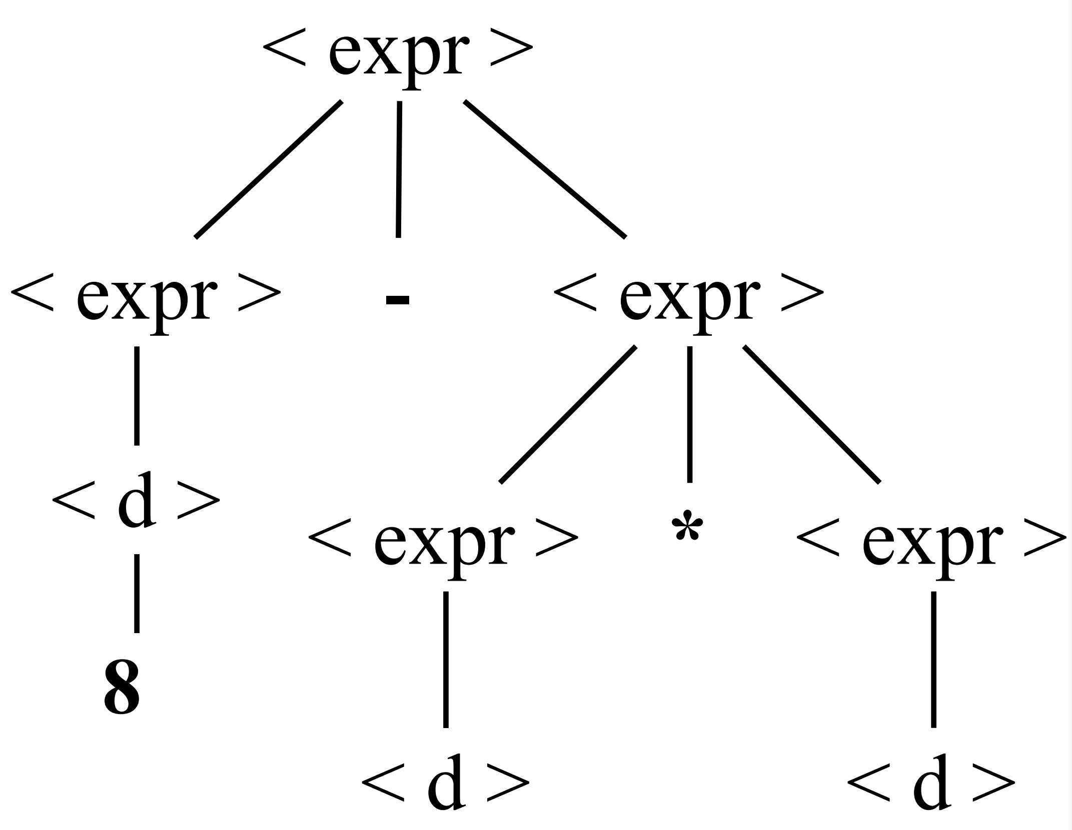


$$
\begin{aligned}
\langle \text{start} \rangle &::= \langle \text{expr} \rangle \\
\langle \text{expr} \rangle &::= \langle \text{expr} \rangle + \langle \text{expr} \rangle \mid \\
&\quad\ \langle \text{expr} \rangle - \langle \text{expr} \rangle \mid \\
&\quad\ \langle \text{expr} \rangle * \langle \text{expr} \rangle \mid \\
&\quad\ \langle \text{expr} \rangle / \langle \text{expr} \rangle \mid \\
&\quad\ \langle \text{expr} \rangle \wedge \langle \text{expr} \rangle \mid \\
&\quad\ \langle \text{d} \rangle \mid \langle \text{l} \rangle \\
\langle \text{d} \rangle &::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \mathbf{9} \\
\langle \text{l} \rangle &::= \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots \mid \mathbf{z}
\end{aligned}
$$

Parse Tree I

Possible Parse Trees

Parse “8 - 3 * 2”:

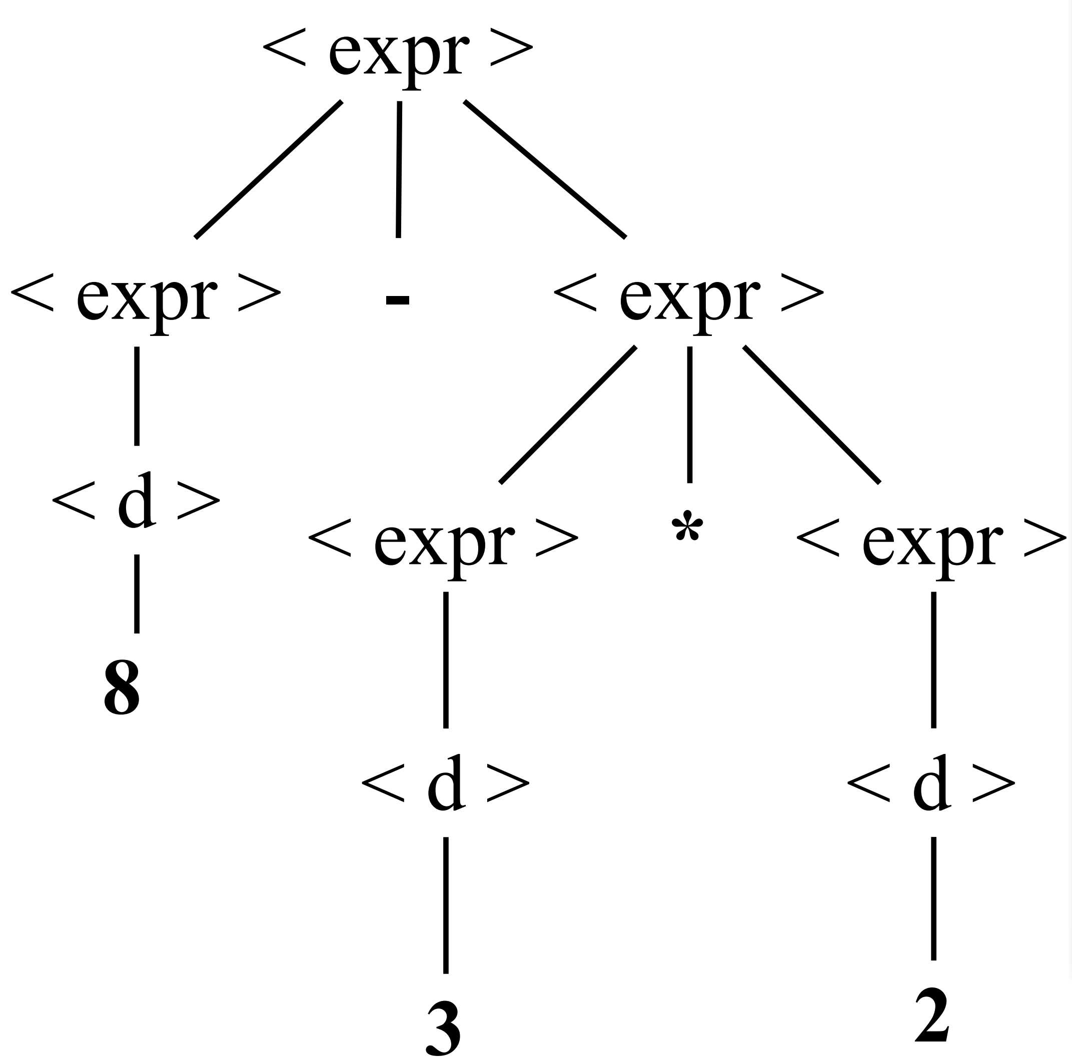


< start > ::= < expr >
< expr > :: = < expr > + < expr > |
< expr > **-** < expr > |
< expr > * < expr > |
< expr > / < expr > |
< expr > ^ < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree I

Possible Parse Trees

Parse "8 - 3 * 2":



< start > ::= < expr >
< expr > :: = < expr > + < expr > |
< expr > - < expr > |
< expr > * < expr > |
< expr > / < expr > |
< expr > ^ < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree I

Possible Parse Trees

Parse “8 - 3 * 2”:

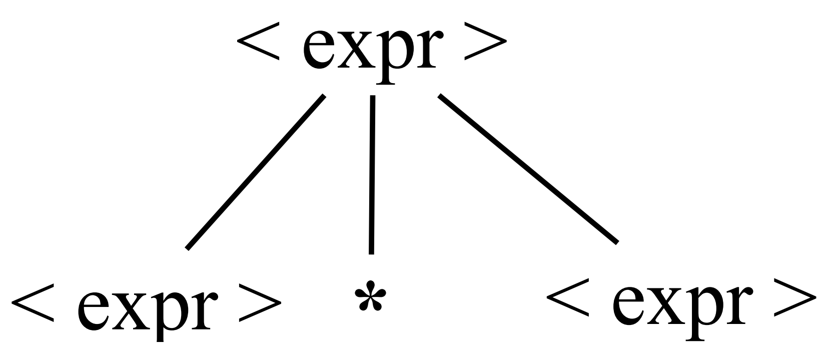
< expr >

< start > ::= < expr >
< expr > :: = < expr > + < expr > |
< expr > **-** < expr > |
< expr > * < expr > |
< expr > / < expr > |
< expr > ^ < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree

Possible Parse Trees

Parse “8 - 3 * 2”:

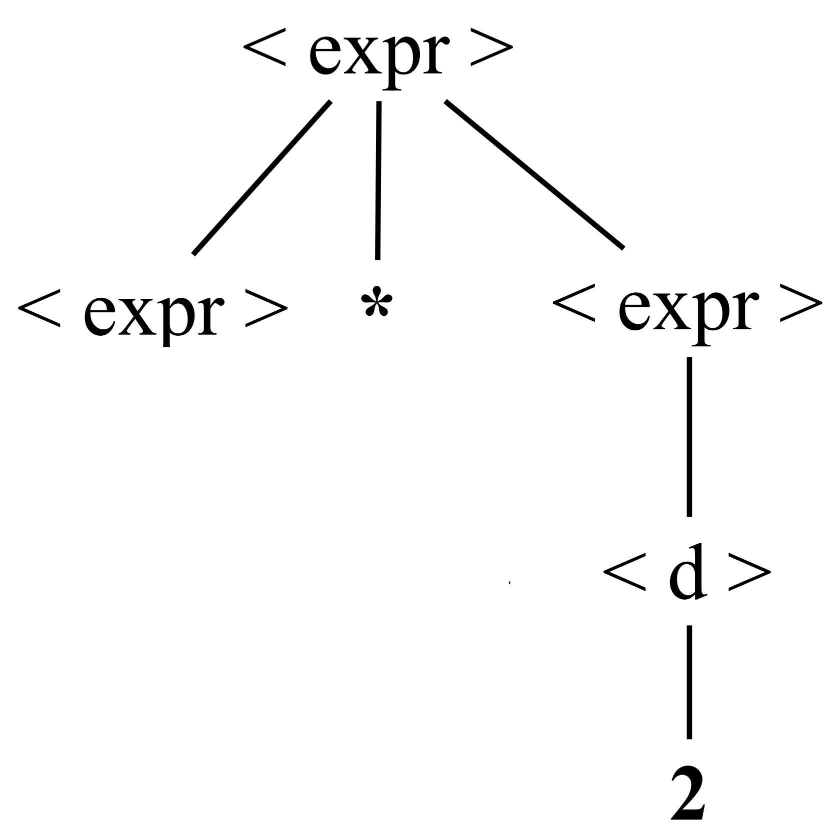


< start > ::= < expr >
< expr > :: = < expr > **+** < expr > |
< expr > **-** < expr > |
< expr > ***** < expr > |
< expr > **/** < expr > |
< expr > **^** < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree

Possible Parse Trees

Parse “8 - 3 * 2”:

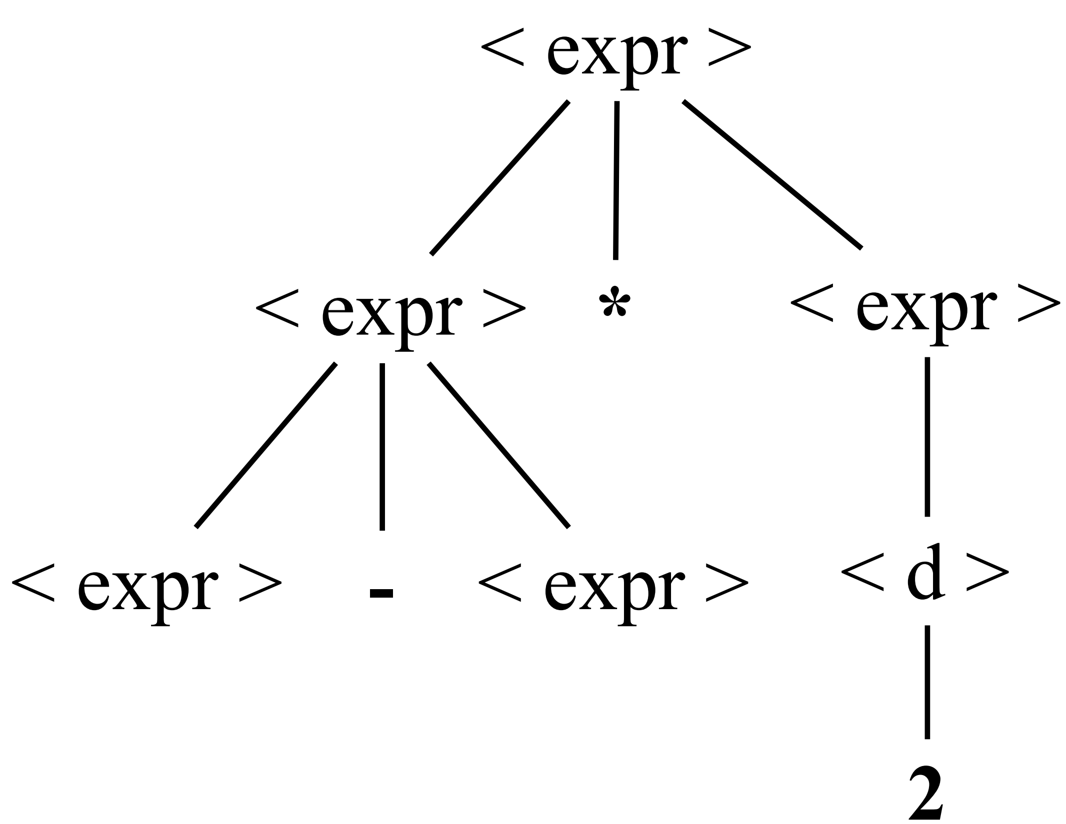


< start > ::= < expr >
< expr > :: = < expr > + < expr > |
< expr > **-** < expr > |
< expr > * < expr > |
< expr > / < expr > |
< expr > ^ < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree

Possible Parse Trees

Parse “8 - 3 * 2”:



< start > ::= < expr >

< expr > :: = < expr > + < expr > |
< expr > **-** < expr > |
< expr > * < expr > |
< expr > / < expr > |
< expr > ^ < expr > |
< d > | < l >

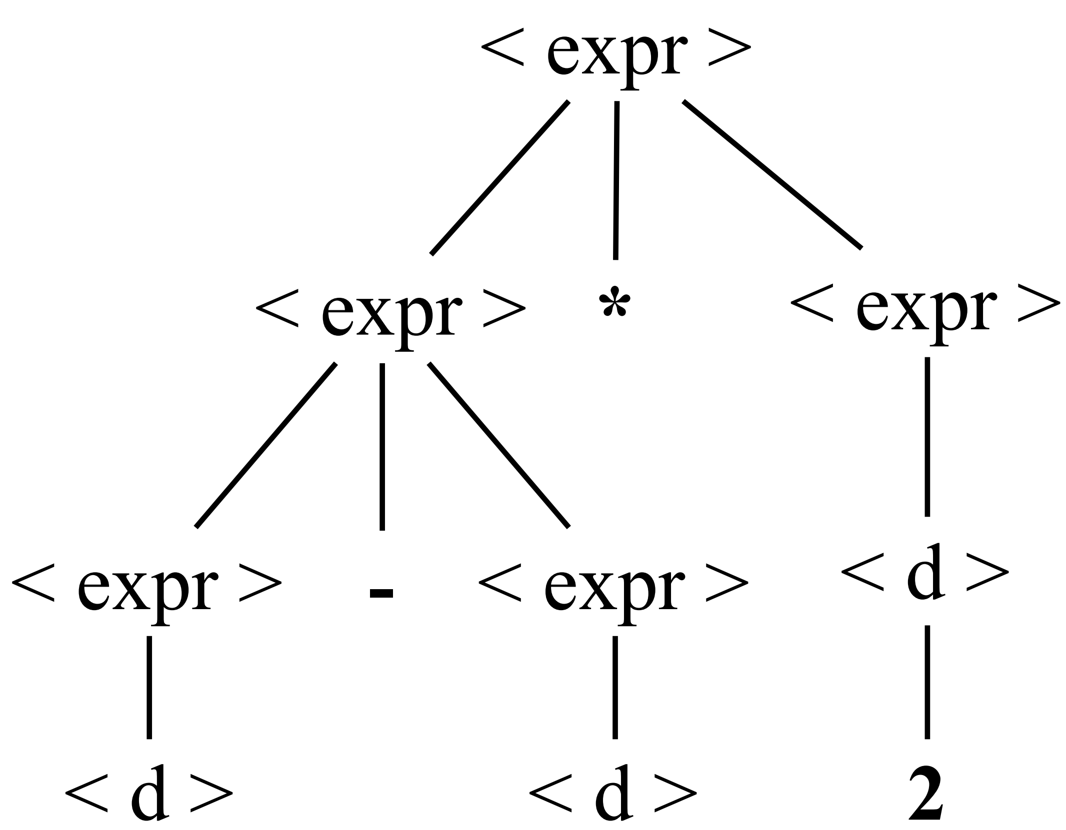
< d > :: = **0** | **1** | **2** | **3** | … | **9**

< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree

Possible Parse Trees

Parse “8 - 3 * 2”:

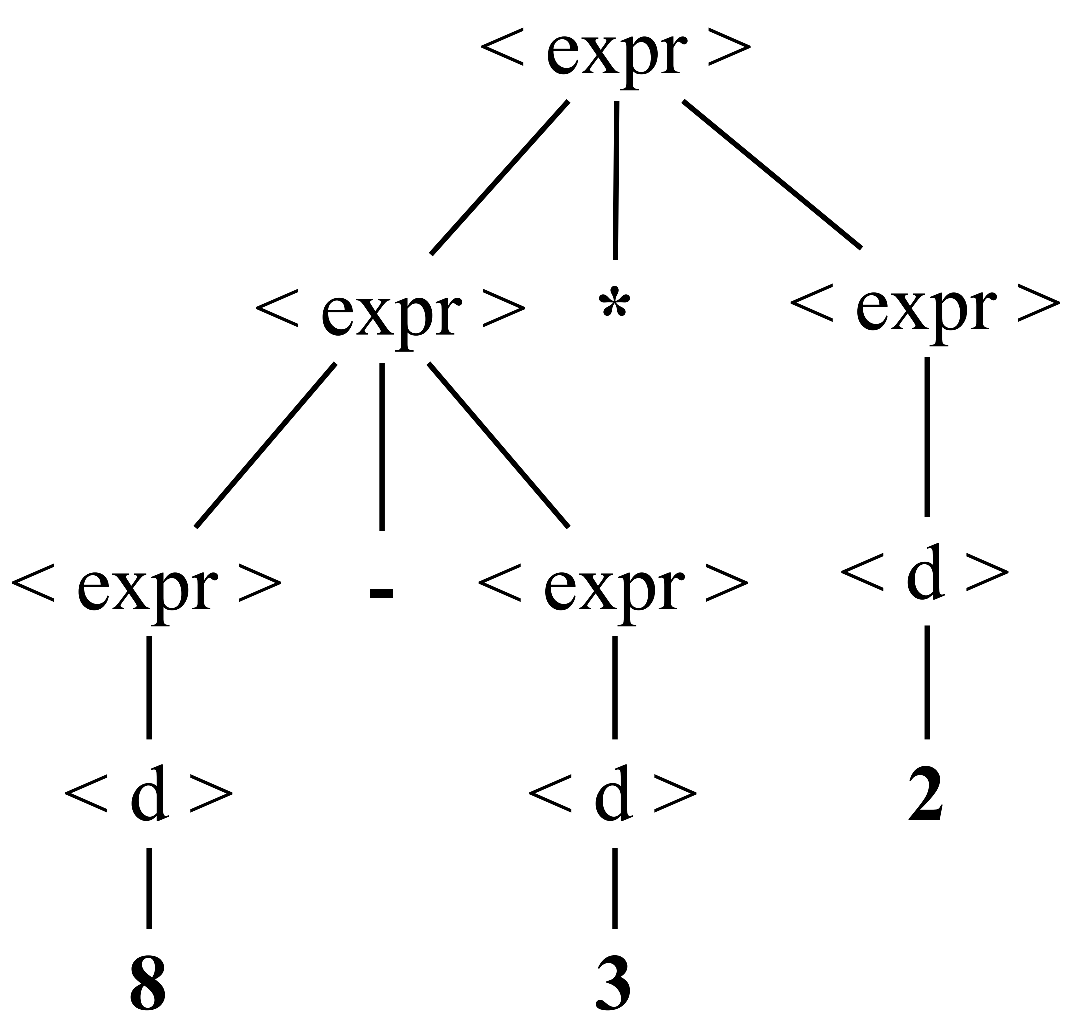


< start > ::= < expr >
< expr > :: = < expr > **+** < expr > |
< expr > **-** < expr > |
< expr > ***** < expr > |
< expr > **/** < expr > |
< expr > **^** < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree

Possible Parse Trees

Parse “8 - 3 * 2”:



< start > ::= < expr >
< expr > :: = < expr > + < expr > |
< expr > - < expr > |
< expr > * < expr > |
< expr > / < expr > |
< expr > ^ < expr > |
< d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Parse Tree

Changing the Language to Include Delimiters

<expr> ::= <expr> − <expr> |
<expr> * <expr> |
<l> | <d>

Changing the Language to Include Delimiters

<expr> ::= (<expr>) − (<expr>) |
(<expr>) * (<expr>) |
<l> | <d>

"8 - 3 * 2"

(8) - ((3) * (2))

OR

((8) - (3)) * (2)

Pretty ugly, isn't it?

Is there any other way to disambiguate our expression grammar?

Changing the Grammar to Impose Precedence

< start > ::= < expr >

< expr > ::= < expr > + < expr > |
< expr > - < expr > |
< term >

< term > ::= < term > * < term > |
< term > / < term > |
< d > | < l >

< d > ::= **0** | **1** | **2** | **3** | … | **9**

< l > ::= **a** | **b** | **c** | … | **z**

New Grammar $\mathcal{G}'$

< start > ::= < expr >

< expr > ::= < expr > + < expr > |
< expr > - < expr > |
< expr > * < expr > |
< expr > / < expr > |
< d > | < l >

< d > ::= **0** | **1** | **2** | **3** | … | **9**

< l > ::= **a** | **b** | **c** | … | **z**

Original Grammar $\mathcal{G}$

Grouping in Parse Tree Now Reflects Precedence

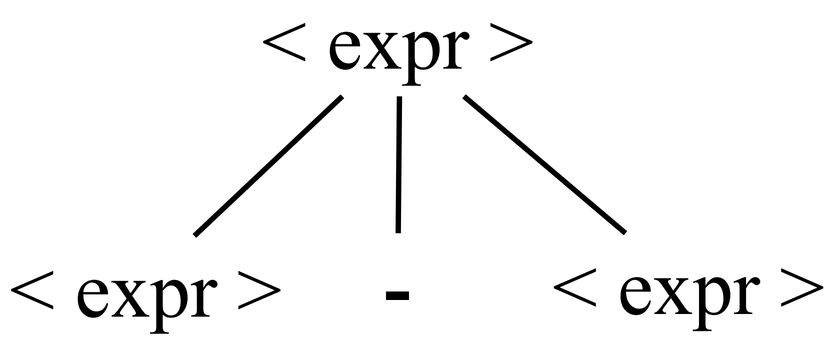
Parse “8 - 3 * 2”:

$$\langle \text{expr} \rangle$$

$$\begin{aligned}
\langle \text{start} \rangle &::= \langle \text{expr} \rangle \\
\langle \text{expr} \rangle &::= \langle \text{expr} \rangle + \langle \text{expr} \rangle \mid \\
&\quad\ \langle \text{expr} \rangle - \langle \text{expr} \rangle \mid \\
&\quad\ \langle \text{term} \rangle \\
\langle \text{term} \rangle &::= \langle \text{term} \rangle * \langle \text{term} \rangle \mid \\
&\quad\ \langle \text{term} \rangle / \langle \text{term} \rangle \mid \\
&\quad\ \langle \text{d} \rangle \mid \langle \text{l} \rangle \\
\langle \text{d} \rangle &::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \mathbf{9} \\
\langle \text{l} \rangle &::= \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots \mid \mathbf{z}
\end{aligned}$$

Grouping in Parse Tree Now Reflects Precedence

Parse “8 - 3 * 2”:



< start > ::= < expr >

< expr > :: = < expr > + < expr > |
< expr > - < expr > |
< term >

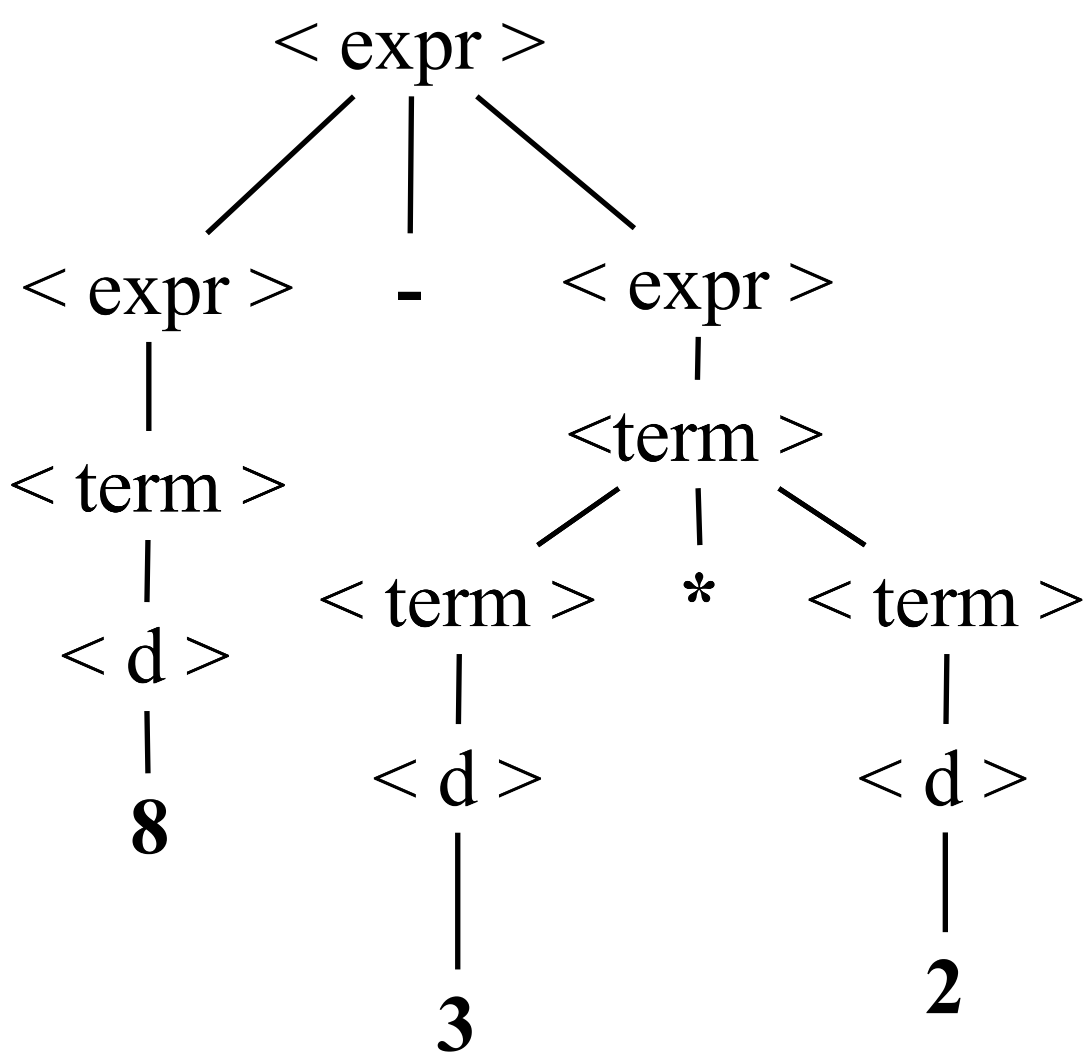
< term > ::= < term > * < term > |
< term > / < term > |
< d > | < l >

< d > :: = **0** | **1** | **2** | **3** | … | **9**

< l > :: = **a** | **b** | **c** | … | **z**

Grouping in Parse Tree Now Reflects Precedence

Parse “8 - 3 * 2”:



< start > ::= < expr >

< expr > :: = < expr > + < expr > |
< expr > - < expr > |
< term >

< term > ::= < term > * < term > |
< term > / < term > |
< d > | < l >

< d > :: = **0** | **1** | **2** | **3** | … | **9**

< l > :: = **a** | **b** | **c** | … | **z**

<u>Only One Possible Parse Tree</u>

Precedence

- *Low Precedence:*
  Addition + and Subtraction -
- *Medium Precedence:*
  Multiplication * and Division /
- *Highest Precedence:*
  Exponentiation ^

```
< start > ::=  < expr >
< expr > :: = < expr > + < expr > |
              < expr > - < expr > |
              < term >
< term > ::=  < term > * < term > |
              < term > / < term > |
              < d > | < l >
< d >      :: = 0 | 1 | 2 | 3 | … | 9
< l >      :: = a | b | c | … | z
```

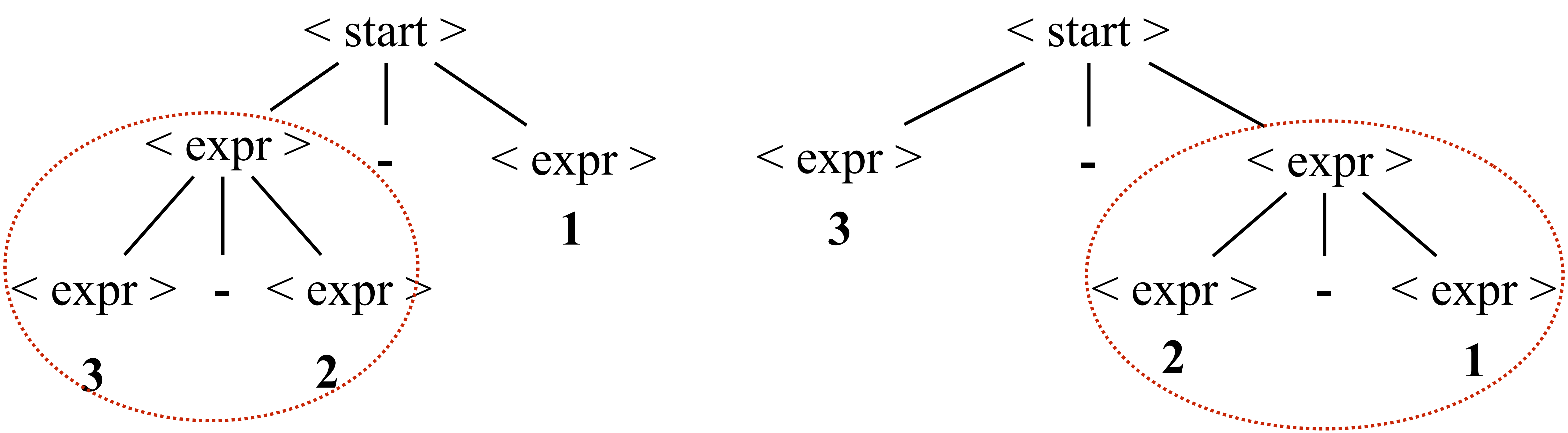
Still Have Ambiguity…

3 - 2 - 1 still a problem:

< start > ::= < expr >
< expr > :: = < expr > + < expr > | **< expr > - < expr >** | < term >
< term > ::= < term > * < term > | < term > / < term > | < d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Still Have Ambiguity…

3 - 2 - 1 still a problem:



Eval: (3 - 2) - 1 = 0 **Eval: 3 - (2 - 1) = 2**

< start > ::= < expr >
< expr > :: = < expr > + < expr > | **< expr > - < expr >** | < term >
< term > ::= < term > * < term > | < term > / < term > | < d > | < l >
< d > :: = **0** | **1** | **2** | **3** | … | **9**
< l > :: = **a** | **b** | **c** | … | **z**

Still Have Ambiguity…

3 - 2 - 1 still a problem:

< start > ::= < expr >

< expr > :: = < expr > + < expr > | < expr > - < expr > | < term >

< term > ::= < term > * < term > | < term > / < term > | < d > | < l >

< d > :: = **0** | **1** | **2** | **3** | … | **9**

< l > :: = **a** | **b** | **c** | … | **z**

- Grouping of operators of same precedence not disambiguated.
- Non-commutative operators: only one parse tree correct.

Imposing Associativity

Same grammar with left / right recursion for - :

Our choices:

$$
\begin{aligned}
<\text{expr}> &::= <\text{d}> \ - \ <\text{expr}> \mid \\
&\quad\ <\text{d}> \\
<\text{d}> &::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \mathbf{9}
\end{aligned}
$$

Or:

$$
\begin{aligned}
<\text{expr}> &::= <\text{expr}> - <\text{d}> \mid \\
&\quad\ <\text{d}> \\
<\text{d}> &::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \mathbf{9}
\end{aligned}
$$

Which one do we want for 3 - 2 - 1 ?

Associativity

- Deals with operators of same precedence
- Implicit grouping or parenthesizing
- Left to right: *, /, +, -
- Right to left: ^

Complete, Unambiguous Arithmetic Expression Grammar

$$
\begin{array}{lll}
< \text{start} > & ::= & < \text{expr} > \\
< \text{expr} > & ::= & < \text{expr} > + < \text{expr} > \mid \\
 & & < \text{expr} > \mathbf{-} < \text{expr} > \mid \\
 & & < \text{expr} > * < \text{expr} > \mid \\
 & & < \text{expr} > / < \text{expr} > \mid \\
 & & < \text{expr} > \wedge < \text{expr} > \mid \\
 & & < \text{d} > \mid < \text{l} > \\
< \text{d} > & ::= & \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \mathbf{9} \\
< \text{l} > & ::= & \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots \mid \mathbf{z}
\end{array}
$$

Original Ambiguous Grammar $\mathcal{G}$

$$
\begin{array}{lll}
< \text{start} > & ::= & < \text{expr} > \\
< \text{expr} > & ::= & < \text{expr} > + < \text{term} > \mid \\
 & & < \text{expr} > - < \text{term} > \mid \\
 & & < \text{term} > \\
< \text{term} > & ::= & < \text{term} > * < \text{factor} > \mid \\
 & & < \text{term} > / < \text{factor} > \mid \\
 & & < \text{factor} > \\
< \text{factor} > & ::= & < \text{g} > \wedge < \text{factor} > \mid \\
 & & < \text{g} > \\
< \text{g} > & ::= & \mathbf{(} < \text{expr} > \mathbf{)} \mid < \text{d} > \mid < \text{l} > \\
< \text{d} > & ::= & \mathbf{0} \mid \mathbf{1} \mid \mathbf{2} \mid \ldots \mid \mathbf{9} \\
< \text{l} > & ::= & \mathbf{a} \mid \mathbf{b} \mid \mathbf{c} \mid \ldots \mid \mathbf{z}
\end{array}
$$

Unambiguous Grammar $\mathcal{G}$

Dealing with Ambiguity

Some facts:

- Can’t always remove an ambiguity from a grammar by restructuring productions.
- An inherently ambiguous language does not possess an unambiguous grammar.
- Detecting ambiguity in context-free grammars is an undecidable problem. There is no polynomial-time algorithm that can examine an arbitrary context-free grammar and tell if it is ambiguous.

Next Lecture

Things to do:

- Read Scott, Chapter 2.2 - 2.5 (skip 2.3.3 bottom-up parsing)