
LL Parsing
Talen & Compilers

Utrecht University

The midterm results

…

Today

LL parsing

Lecture notes: 9.1, 10.1

(Note: I added a chapter before LL parsing.)

Learning goals

- Parse a sentence top-down using the LL(1) algorithm
- Perform grammar analyses to determine whether or not a grammar is LL(1): determine empty, first, follow, and lookAhead

Parsing until now

Parser combinators: can parse sentences from many grammars and are often efficient. Can be slow for ambiguous grammars or grammars that can be left-factored

(N)DFAs: can parse regular languages in linear time

LR parsing: can parse sentences from LR grammars in linear time

Earley parsing/CKY: can parse sentences from any grammar in time $O(n^3)$.

Linear-time parsing

When developing a compiler, we want parsing to happen in linear time

LL & LR parsing are potential solutions

LR parsing:

- Left-to-right input processing
- Rightmost derivation
- Bottom-up

LL parsing:

- Left-to-right input processing
- Leftmost derivation
- Top-down

LL parsing

LL parsing uses a state:

- a stack containing a sequence of terminals and nonterminals
- the current input

Start with (S,input)

At each step:

- **Expand**: top nonterminal symbol by a right-hand side
- **Match**: top terminal symbol with first symbol in input

Succeed if stack and input are empty, otherwise fail

Utrecht University

Example I

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
S	aab	

Example I

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
S	aab	Expand
cS	aab	

Utrecht University

Example I

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
S	aab	Expand
cS	aab	Fail

Utrecht University

Example I

Grammar: S → aS | cS | b   Input: aab

Stack	Input	Action
S	aab	Expand
aS	aab	

Utrecht University

Example I

Grammar: S → aS | cS | b   Input: aab

Stack	Input	Action
S	aab	Expand
aS	aab	Match
S	ab	

Utrecht University

Example I

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
S	aab	Expand
aS	aab	Match
S	ab	Expand
aS	ab	

Utrecht University

Example I

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
S	aab	Expand
aS	aab	Match
S	ab	Expand
aS	ab	Match
S	b	

Utrecht University

Example I

Grammar: $S \rightarrow aS \mid cS \mid b$ Input: aab

Stack	Input	Action
S	aab	Expand
aS	aab	Match
S	ab	Expand
aS	ab	Match
S	b	Expand
b	b	

Example I

Grammar: S → aS | cS | b Input: aab

Stack	Input	Action
S	aab	Expand
aS	aab	Match
S	ab	Expand
aS	ab	Match
S	b	Expand
b	b	Match
ε	ε	

Example I

Grammar: $S \rightarrow aS \mid cS \mid b$ Input: aab

Stack	Input	Action
S	aab	Expand
aS	aab	Match
S	ab	Expand
aS	ab	Match
S	b	Expand
b	b	Match
ε	ε	Succeed

Expanding which right-hand side?

Grammar: S → aS | cS | b Input: aab

Look at the **first** symbol a production can derive

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	Match
BC	ccba	

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	Match
BC	ccba	Expand?

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	Match
BC	ccba	Expand
ccC	ccba	

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	Match
BC	ccba	Expand
ccC	ccba	Match
cC	cba	

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	Match
BC	ccba	Expand
ccC	ccba	Match
cC	cba	Match
C	ba	

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	Match
BC	ccba	Expand
ccC	ccba	Match
cC	cba	Match
C	ba	Expand
ba	ba	

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	Match
BC	ccba	Expand
ccC	ccba	Match
cC	cba	Match
C	ba	Expand
ba	ba	Match
a	a	

Utrecht University

A second example

S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba

Input: ccccba

The **first set** of C consists of a and b

Stack	Input	Action
S	ccccba	Expand
cA	ccccba	Match
A	cccba	Expand
cBC	cccba	Match
BC	ccba	Expand
ccC	ccba	Match
cC	cba	Match
C	ba	Expand
ba	ba	Match
a	a	Match
ε	ε	Succeed

Utrecht University

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	

Utrecht University

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	Expand?

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	Expand
abA	abbb	

Utrecht University

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	Expand
abA	abbb	Match
bA	bbb	

Utrecht University

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	Expand
abA	abbb	Match
bA	bbb	Match
A	bb	

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	Expand
abA	abbb	Match
bA	bbb	Match
A	bb	Expand?

Utrecht University

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	Expand
abA	abbb	Match
bA	bbb	Match
A	bb	Expand
bb	bb	

Utrecht University

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	Expand
abA	abbb	Match
bA	bbb	Match
A	bb	Expand
bb	bb	Match
b	b	

Example 3

S → abA | aa
A → bb | bS

Input: abbb

Stack	Input	Action
S	abbb	Expand
abA	abbb	Match
bA	bbb	Match
A	bb	Expand
bb	bb	Match
b	b	Match
ε	ε	Succeed

We need to look ahead two symbols to choose between the productions from S

We need the first set of S to choose between the productions from A

Utrecht University

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	Expand?

Utrecht University

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	Expand
AaS	acbab	

Utrecht University

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	Expand
AaS	acbab	Expand?

Utrecht University

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	Expand
AaS	acbab	Expand
aS	acbab	

Utrecht University

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	Expand
AaS	acbab	Expand
aS	acbab	Match
S	cbab	

Utrecht University

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	Expand
AaS	acbab	Expand
aS	acbab	Match
S	cbab	Expand
AaS	cbab	

Utrecht University

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	Expand
AaS	acbab	Expand
aS	acbab	Match
S	cbab	Expand
AaS	cbab	Expand
cSaS	cbab	

Example 4

S → AaS | B
A → cS | ε
B → b

Input: acbab

Stack	Input	Action
S	acbab	Expand
AaS	acbab	Expand
aS	acbab	Match
S	cbab	Expand
AaS	cbab	Expand
cSaS	cbab	Match
SaS	bab	Rest is easy

Because A can derive the **empty** string, S → AaS can start with a

Because A can derive the ε, we need to know which symbols can **follow** on an A in a derivation

Utrecht University

LL(1) grammars

The lookahead set of a production `N → α` is defined as:

`lookAhead (N → α) = { x | S =>* γNδ => γαδ =>* γxβ}`

A grammar is LL(1) if all pairs of productions of the same nonterminal have disjoint lookahead sets:

`lookAhead (N → α) ∩ lookAhead (N → β) = ∅`

for all pairs of different productions `N → α, N → β`.

Utrecht University

Example 2 revisited

```
S → cA | b
A → cBC | bSA | a
B → cc | Cb
C → aS | ba
```

```
lookAhead (S → cA)  = {c}
lookAhead (S → b)   = {b}
lookAhead (A → cBC) = {c}
lookAhead (A → bSA) = {b}
lookAhead (A → a)   = {a}
lookAhead (B → cc)  = {c}
lookAhead (B → Cb)  = {a,b}
lookAhead (C → aS)  = {a}
lookAhead (C → ba)  = {b}
```

This grammar is LL(1)

Example 3 revisited

```
S → abA | aa
A → bb | bS
```

```
lookAhead (S → abA) = {a}
lookAhead (S → aa)  = {a}
lookAhead (A → bb)  = {b}
lookAhead (A → bS)  = {b}
```

This grammar is **not** LL(1)

It is LL(2)

Calculating lookahead sets

To calculate lookahead sets, we need to know:

- whether or not a nonterminal can derive the empty string
- which terminals can appear as the first symbol in a string derived from a nonterminal
- which terminals can follow a nonterminal in any derivation

These are **grammar analyses** problems

Utrecht University

Deriving the empty string

A nonterminal N can derive the empty string if $N \Rightarrow^* \varepsilon$.

empty $N = N \Rightarrow^* \varepsilon$

$S \rightarrow AaS \mid B$
$A \rightarrow cS \mid \varepsilon$
$B \rightarrow b$

Deriving empty in n steps

The empty function satisfies:

$$\text{empty}_n\ A = \bigcup_{A \rightarrow \beta} (\text{emptyRhs}_n\ \beta)$$

$$\begin{aligned}
\text{emptyRhs}_n\ \varepsilon &= \text{True} \\
\text{emptyRhs}_n\ (B\beta) &= \text{empty}_{n-1}\ B \wedge \text{emptyRhs}_n\ \beta \\
\text{emptyRhs}_n\ (x\beta) &= \text{False}
\end{aligned}$$

Computing empty

Start with empty A = False for all nonterminals A

{ A | empty_0 A }
{ A | empty_1 A }
{ A | empty_2 A }
…

The first set is empty
The next set includes the previous set
We reach a **fixed point** sooner or later
The fixed point is our result

Example fixed point calculation

$S \rightarrow AaS \mid B$

$A \rightarrow cS \mid \varepsilon$

$B \rightarrow b$

$C \rightarrow AA \mid B$

Utrecht University

1 Go to **wooclap.com**

2 Enter the event code in the top banner

Event code
DDFRMX

Enable answers by SMS

Disable reactions

Utrecht University

Q1

First set

The **first set** of a nonterminal N is the set of terminals that can appear as the first symbol of a string derived from N:

$$\text{first } N = \{ x \mid N \Rightarrow^* x\beta \}$$

$S \rightarrow AaS \mid B$
$A \rightarrow cS \mid \varepsilon$
$B \rightarrow b$

first S = {a,b,c}
first A = {c}
first B = {b}

Deriving first sets in n steps

The first function satisfies:

$\text{first}_0\ A = \emptyset$
$\text{first}_n\ A = \bigcup_{A\to\beta} (\text{firstRhs}_n\ \beta)$

$\text{firstRhs}_n\ \varepsilon \quad = \emptyset$
$\text{firstRhs}_n\ (B\beta) = \text{first}_{n-1}\ B \cup \text{if empty } B \text{ then } \text{firstRhs}_n\ \beta \text{ else } \emptyset$
$\text{firstRhs}_n\ (x\beta) = \{x\}$

The sequence $\{ (A,\text{first}_n\ A) \}$ reaches a fixed point

Example fixed point calculation

$S \rightarrow AaS \mid B$

$A \rightarrow cS \mid \varepsilon$

$B \rightarrow b$

$C \rightarrow AA \mid B$

Q2

Utrecht University

Follow set

The **follow set** of a nonterminal N is the set of terminals that can follow N in any derivation starting with the start symbol S:

follow N = $\{ x \mid S \Rightarrow^* \alpha N x \beta \}$

$S \rightarrow AaS \mid B$	follow S = {a}
$A \rightarrow cS \mid \varepsilon$	follow A = {a}
$B \rightarrow b$	follow B = {a}

Deriving follow sets in n steps

The follow function satisfies:

$\text{follow}_0\ A = \emptyset$
$\text{follow}_n\ A = \bigcup_{B \rightarrow \alpha A \beta} (\text{followRhs}_n\ B\ \beta)$

$\text{followRhs}_n\ B\ \beta = \text{firstRhs}\ \beta \cup \text{if emptyRhs}\ \beta \text{ then follow}_{n-1}\ B \text{ else } \emptyset$

The sequence $\{\ (A,\text{follow}_n\ A)\ \}$ reaches a fixed point

Utrecht University

Example fixed point calculation

S → AaS | B

A → cS | ε

B → b

C → AA | B

Utrecht University

Q3,4

Computing lookAhead

lookAhead (A → β) = followRhs A β

Summary

Just as LR, top-down parsing using LL(1) uses a stack for parsing

To determine which production to use when expanding a nonterminal, LL(1) uses the lookAhead sets of productions

To determine the lookAhead set of a production we use grammar analyses such as empty, first and follow

Grammar analyses are calculated using fixed points calculations