



Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Talen en Compilers

2019 - 2020, period 2

Jurriaan Hage

Department of Information and Computing Sciences
Utrecht University

2021-01-18

13. LL parsing



This lecture

LL parsing

Motivation

Good expansion choices

LL(1) grammars

Computing empty, first, and follow



13.1 Motivation



Time complexity of parsing

- ▶ Earley's algorithm can parse **any** context-free language. It runs in $O(n^3)$ where n is the length of the input. (It behaves better on many important special cases.)
- ▶ Our parser combinators can parse a large class of context-free languages, but no left-recursive grammars. They are fast for many important classes of grammars, but **slow** for ambiguous grammars or grammars that are not left-factored.
- ▶ A DFA can parse any **regular** language in linear time.



The big question

Are there more languages that we can parse in **linear** time?



The big question

Are there more languages that we can parse in
linear time?

Of course!



Expand-Match parsing algorithm

We start with stack containing only S where S is the start symbol. Then in each step, we do one of the following:

- Expand** If the top symbol on the stack is a nonterminal, we replace it with a corresponding right hand side of a production in a **non-deterministic** fashion.
- Match** If the top symbol on the stack is a terminal and the first symbol of the input matches, we **consume** the symbol from the input and **remove** it from the stack.

If both stack and input are empty, we succeed. In any other case, we signal an error.



Example

| $S \rightarrow aS \mid cS \mid b$

Let us parse aab:

stack input

S aab initial state – expand, but which production?



Example

| $S \rightarrow aS \mid cS \mid b$

Let us parse aab:

stack input

 S aab

 cS aab one option – match fails, thus error



Example

| $S \rightarrow aS \mid cS \mid b$

Let us parse aab:

stack input

 S aab

 aS aab other option – match succeeds



Example

$$S \rightarrow aS \mid cS \mid b$$

Let us parse aab:

stack input

S aab

aS aab

S ab expand again



Example

| $S \rightarrow aS \mid cS \mid b$

Let us parse aab:

stack input

S aab

aS aab

S ab

aS ab match succeeds



Example

| $S \rightarrow aS \mid cS \mid b$

Let us parse aab:

stack	input	
S	aab	
aS	aab	
S	ab	
aS	ab	
S	b	expand



Example

| $S \rightarrow aS \mid cS \mid b$

Let us parse aab:

stack	input	
S	aab	
aS	aab	
S	ab	
aS	ab	
S	b	
b	b	match



Example

$S \rightarrow aS \mid cS \mid b$

Let us parse aab:

stack input

S aab

aS aab

S ab

aS ab

S b

b b

ϵ ϵ

parse successful



LL stack machines

This kind of automata are also called **LL (stack) machines**

- ▶ The machine traverses the input starting from the **left**,
- ▶ The machine produces a **leftmost** derivation.



LL stack machines

This kind of automata are also called **LL (stack) machines**

- ▶ The machine traverses the input starting from the **left**,
- ▶ The machine produces a **leftmost** derivation.

The LL machine is nondeterministic:

- ▶ in every expand step, we have to choose one of the productions for the given nonterminal,
- ▶ the choice can have an effect on whether the parse succeeds or not.

An LL machine **accepts** a word if there exists a sequence of choices leads to success (similar to NFA acceptance).



13.2 Good expansion choices



Idea

By looking at the grammar, let us try to determine which choices are wrong, and which may not be.



Idea

By looking at the grammar, let us try to determine which choices are wrong, and which may not be.

For example, recall:

$$S \rightarrow aS \mid cS \mid b$$

We started at:

stack	input
S	aab

By looking at the first symbol in the input (a), we know that only the production aS may lead to success.



Idea

By looking at the grammar, let us try to determine which choices are wrong, and which may not be.

For example, recall:

$$S \rightarrow aS \mid cS \mid b$$

We started at:

stack	input
S	aab

By looking at the first symbol in the input (a), we know that only the production aS may lead to success.

Can we do something similar for other grammars?



Example 1

S ccccba initial state, expand

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

Let us parse ccccba.



Example 1

S cccba
cA cccba match

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

Let us parse cccba.



Example 1

S cccba
cA cccba
A ccba expand

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

Let us parse cccba.



Example 1

$S \rightarrow cA \mid b$
 $A \rightarrow cBC \mid bSA \mid a$
 $B \rightarrow cc \mid Cb$
 $C \rightarrow aS \mid ba$

S cccba
cA cccba
A ccba
cBC ccba match

Let us parse cccba.



Example 1

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

S cccba
cA cccba
A ccba
cBC cccba
BC ccba

expand, but how?

Let us parse cccba.



Example 1

	S	ccccba	
	cA	ccccba	
	A	cccba	
	cBC	cccba	
	BC	ccba	expand, but how?
	$S \rightarrow cA \mid b$		
	$A \rightarrow cBC \mid bSA \mid a$		
	$B \rightarrow cc \mid Cb$		
	$C \rightarrow aS \mid ba$		

Let us parse cccba.

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



Example 1

	S	ccccba	
	cA	ccccba	
	A	cccba	
	cBC	cccba	
	BC	ccba	
	ccC	ccba	match
	$S \rightarrow cA \mid b$		
	$A \rightarrow cBC \mid bSA \mid a$		
	$B \rightarrow cc \mid Cb$		
	$C \rightarrow aS \mid ba$		

Let us parse cccba.

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



Example 1

	S	ccccba	
	cA	ccccba	
	A	cccba	
	cBC	cccba	
	BC	ccba	
	ccC	ccba	
	cC	cba	match
S	→	cA b	
A	→	cBC bSA a	
B	→	cc Cb	
C	→	aS ba	

Let us parse ccccba.

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



Example 1

	S	ccccba	
	cA	ccccba	
	A	cccba	
	cBC	cccba	
	BC	ccba	
	ccC	ccba	
	cC	cba	
	C	ba	expand

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

Let us parse ccccba.

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



Example 1

	S	ccccba	
	cA	ccccba	
	A	cccba	
	cBC	cccba	
	BC	ccba	
	ccC	ccba	
	cC	cba	
	C	ba	
	ba	ba	match the rest

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

Let us parse ccccba.

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



Example 1

	S	ccccba
	cA	ccccba
	A	cccba
	cBC	cccba
	BC	ccba
	ccC	ccba
	cC	cba
	C	ba
	ba	ba
	a	a
	ϵ	ϵ

$S \rightarrow cA \mid b$
 $A \rightarrow cBC \mid bSA \mid a$
 $B \rightarrow cc \mid Cb$
 $C \rightarrow aS \mid ba$

Let us parse cccba.

parse successful

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



Example 2

S abbb expand, but how?

$$\begin{array}{l} S \rightarrow abA \mid aa \\ A \rightarrow bb \mid bS \end{array}$$

Let us parse abbb.



Example 2

S abbb expand, but how?

$$\begin{array}{l} S \rightarrow abA \mid aa \\ A \rightarrow bb \mid bS \end{array}$$

Let us parse abbb.

- ▶ The first symbol of both productions for S is a, we need the first **two** symbols to resolve the choice.



Example 2

S abbb
abA abbb match twice

$S \rightarrow abA \mid aa$
 $A \rightarrow bb \mid bS$

Let us parse abbb.

- ▶ The first symbol of both productions for S is a, we need the first **two** symbols to resolve the choice.



Example 2

	S	abbb	
	abA	abbb	
	S → abA aa	bA	bbb
	A → bb bS	A	bb

expand, but how?

Let us parse abbb.

- ▶ The first symbol of both productions for S is a, we need the first **two** symbols to resolve the choice.



Example 2

	S	abbb	
	abA	abbb	
	S → abA aa	bA	bbb
	A → bb bS	A	bb

expand, but how?

Let us parse abbb.

- ▶ The first symbol of both productions for S is a, we need the first **two** symbols to resolve the choice.
- ▶ We again need the first set of S to resolve the choice for A.



Example 2

	S	abbb		
	abA	abbb		
	$S \rightarrow abA \mid aa$	bA	bbb	
	$A \rightarrow bb \mid bS$	A	bb	
		bb	bb	match the rest
	Let us parse abbb.			

- ▶ The first symbol of both productions for S is a, we need the first **two** symbols to resolve the choice.
- ▶ We again need the first set of S to resolve the choice for A.



Example 2

	S	abbb	
	abA	abbb	
	$S \rightarrow abA \mid aa$	bA	bbb
	$A \rightarrow bb \mid bS$	A	bb
	bb	bb	
Let us parse abbb.	b	b	
	ϵ	ϵ	parse successful

- ▶ The first symbol of both productions for S is a, we need the first **two** symbols to resolve the choice.
- ▶ We again need the first set of S to resolve the choice for A.



Example 3

S acbab expand, but how?

$$\begin{array}{l} S \rightarrow AaS \mid B \\ A \rightarrow cS \mid \varepsilon \\ B \rightarrow b \end{array}$$

Let us parse acbab.



Example 3

S acbab expand, but how?

$$\begin{array}{l} S \rightarrow AaS \mid B \\ A \rightarrow cS \mid \varepsilon \\ B \rightarrow b \end{array}$$

Let us parse acbab.

- ▶ Because A can derive the **empty** string, the production $S \rightarrow AaS$ can start with a.



Example 3

$$\begin{array}{l} S \rightarrow AaS \mid B \\ A \rightarrow cS \mid \varepsilon \\ B \rightarrow b \end{array}$$

S acbab
AaS acbab expand, but how?

Let us parse acbab.

- ▶ Because A can derive the **empty** string, the production $S \rightarrow AaS$ can start with a.



Example 3

$$\begin{array}{l} S \rightarrow AaS \mid B \\ A \rightarrow cS \mid \varepsilon \\ B \rightarrow b \end{array}$$

S acbab
AaS acbab expand, but how?

Let us parse acbab.

- ▶ Because A can derive the **empty** string, the production $S \rightarrow AaS$ can start with a.
- ▶ Because A can derive ε , we have to know which symbols can **follow** on an A in a derivation.



Example 3

	S	acbab	
	AaS	acbab	
	aS	acbab	match
	$S \rightarrow AaS \mid B$		
	$A \rightarrow cS \mid \varepsilon$		
	$B \rightarrow b$		

Let us parse acbab.

- ▶ Because A can derive the **empty** string, the production $S \rightarrow AaS$ can start with a.
- ▶ Because A can derive ε , we have to know which symbols can **follow** on an A in a derivation.



Example 3

	S	acbab	
	AaS	acbab	
	aS	acbab	
	S	cbab	expand
	$S \rightarrow AaS \mid B$		
	$A \rightarrow cS \mid \varepsilon$		
	$B \rightarrow b$		

Let us parse acbab.

- ▶ Because A can derive the **empty** string, the production $S \rightarrow AaS$ can start with a.
- ▶ Because A can derive ε , we have to know which symbols can **follow** on an A in a derivation.



Example 3

	$S \rightarrow AaS \mid B$	S	acbab	
	$A \rightarrow cS \mid \varepsilon$	AaS	acbab	
	$B \rightarrow b$	aS	acbab	
		S	cbab	
		AaS	cbab	expand

Let us parse acbab.

- ▶ Because A can derive the **empty** string, the production $S \rightarrow AaS$ can start with a.
- ▶ Because A can derive ε , we have to know which symbols can **follow** on an A in a derivation.



Example 3

	$S \rightarrow AaS \mid B$	S	acbab	
	$A \rightarrow cS \mid \varepsilon$	AaS	acbab	
	$B \rightarrow b$	aS	acbab	
		S	cbab	
		AaS	cbab	
		cSaS	cbab	match

Let us parse acbab.

- ▶ Because A can derive the **empty** string, the production $S \rightarrow AaS$ can start with a.
- ▶ Because A can derive ε , we have to know which symbols can **follow** on an A in a derivation.



Example 3

	S	acbab	
	AaS	acbab	
	$S \rightarrow AaS \mid B$	aS	acbab
	$A \rightarrow cS \mid \varepsilon$	S	cbab
	$B \rightarrow b$	AaS	cbab
		cSaS	cbab
Let us parse acbab.	SaS	bab	rest is easy

- ▶ Because A can derive the **empty** string, the production $S \rightarrow AaS$ can start with a.
- ▶ Because A can derive ε , we have to know which symbols can **follow** on an A in a derivation.



13.3 LL(1) grammars



Definition of LL(1) grammars

Definition

The **lookahead set** of a production $N \rightarrow \alpha$ is defined as

$$\text{lookAhead}(N \rightarrow \alpha) = \{x \mid S \Rightarrow^* \gamma N \delta \Rightarrow \gamma \alpha \delta \Rightarrow^* \gamma x \beta\}$$



Definition of LL(1) grammars

Definition

The **lookahead set** of a production $N \rightarrow \alpha$ is defined as

$$\text{lookAhead}(N \rightarrow \alpha) = \{x \mid S \Rightarrow^* \gamma N \delta \Rightarrow \gamma \alpha \delta \Rightarrow^* \gamma x \beta\}$$

Definition

A grammar is called **LL(1)** if all pairs of productions of the same nonterminal have disjoint lookahead sets, i.e., if

$$\text{lookAhead}(N \rightarrow \alpha) \cap \text{lookAhead}(N \rightarrow \beta) = \emptyset$$

for all pairs of different productions $N \rightarrow \alpha$, $N \rightarrow \beta$.



Example 1 revisited

Before we learn how to compute lookAhead systematically, some simple examples:



Example 1 revisited

Before we learn how to compute lookAhead systematically, some simple examples:

$S \rightarrow cA \mid b$

$A \rightarrow cBC \mid bSA \mid a$

$B \rightarrow cc \mid Cb$

$C \rightarrow aS \mid ba$



Example 1 revisited

Before we learn how to compute lookAhead systematically, some simple examples:

$$\begin{array}{l} S \rightarrow cA \mid b \\ A \rightarrow cBC \mid bSA \mid a \\ B \rightarrow cc \mid Cb \\ C \rightarrow aS \mid ba \end{array}$$
$$\begin{array}{l} \text{lookAhead } (S \rightarrow cA) = \\ \text{lookAhead } (S \rightarrow b) = \\ \text{lookAhead } (A \rightarrow cBC) = \\ \text{lookAhead } (A \rightarrow bSA) = \\ \text{lookAhead } (A \rightarrow a) = \\ \text{lookAhead } (B \rightarrow cc) = \\ \text{lookAhead } (B \rightarrow Cb) = \\ \text{lookAhead } (C \rightarrow aS) = \\ \text{lookAhead } (C \rightarrow ba) = \end{array}$$


Example 1 revisited

Before we learn how to compute lookAhead systematically, some simple examples:

$$\begin{array}{l} S \rightarrow cA \mid b \\ A \rightarrow cBC \mid bSA \mid a \\ B \rightarrow cc \mid Cb \\ C \rightarrow aS \mid ba \end{array}$$
$$\begin{array}{l} \text{lookAhead}(S \rightarrow cA) = \{c\} \\ \text{lookAhead}(S \rightarrow b) = \{b\} \\ \text{lookAhead}(A \rightarrow cBC) = \{c\} \\ \text{lookAhead}(A \rightarrow bSA) = \{b\} \\ \text{lookAhead}(A \rightarrow a) = \{a\} \\ \text{lookAhead}(B \rightarrow cc) = \{c\} \\ \text{lookAhead}(B \rightarrow Cb) = \\ \text{lookAhead}(C \rightarrow aS) = \{a\} \\ \text{lookAhead}(C \rightarrow ba) = \{b\} \end{array}$$


Example 1 revisited

Before we learn how to compute lookAhead systematically, some simple examples:

$$\begin{array}{l} S \rightarrow cA \mid b \\ A \rightarrow cBC \mid bSA \mid a \\ B \rightarrow cc \mid Cb \\ C \rightarrow aS \mid ba \end{array}$$
$$\begin{array}{l} \text{lookAhead}(S \rightarrow cA) = \{c\} \\ \text{lookAhead}(S \rightarrow b) = \{b\} \\ \text{lookAhead}(A \rightarrow cBC) = \{c\} \\ \text{lookAhead}(A \rightarrow bSA) = \{b\} \\ \text{lookAhead}(A \rightarrow a) = \{a\} \\ \text{lookAhead}(B \rightarrow cc) = \{c\} \\ \text{lookAhead}(B \rightarrow Cb) = \{a, b\} \\ \text{lookAhead}(C \rightarrow aS) = \{a\} \\ \text{lookAhead}(C \rightarrow ba) = \{b\} \end{array}$$


Example 1 revisited

Before we learn how to compute lookAhead systematically, some simple examples:

$$\begin{array}{l} S \rightarrow cA \mid b \\ A \rightarrow cBC \mid bSA \mid a \\ B \rightarrow cc \mid Cb \\ C \rightarrow aS \mid ba \end{array}$$
$$\begin{array}{l} \text{lookAhead}(S \rightarrow cA) = \{c\} \\ \text{lookAhead}(S \rightarrow b) = \{b\} \\ \text{lookAhead}(A \rightarrow cBC) = \{c\} \\ \text{lookAhead}(A \rightarrow bSA) = \{b\} \\ \text{lookAhead}(A \rightarrow a) = \{a\} \\ \text{lookAhead}(B \rightarrow cc) = \{c\} \\ \text{lookAhead}(B \rightarrow Cb) = \{a, b\} \\ \text{lookAhead}(C \rightarrow aS) = \{a\} \\ \text{lookAhead}(C \rightarrow ba) = \{b\} \end{array}$$

This grammar is LL(1).



Example 2 revisited

$$\begin{array}{l} S \rightarrow abA \mid aa \\ A \rightarrow bb \mid bS \end{array}$$


Example 2 revisited

$S \rightarrow abA \mid aa$
 $A \rightarrow bb \mid bS$

$\text{lookAhead}(S \rightarrow abA) =$
 $\text{lookAhead}(S \rightarrow aa) =$
 $\text{lookAhead}(A \rightarrow bb) =$
 $\text{lookAhead}(A \rightarrow bS) =$



Example 2 revisited

$S \rightarrow abA \mid aa$
 $A \rightarrow bb \mid bS$

$\text{lookAhead}(S \rightarrow abA) = \{a\}$

$\text{lookAhead}(S \rightarrow aa) = \{a\}$

$\text{lookAhead}(A \rightarrow bb) = \{b\}$

$\text{lookAhead}(A \rightarrow bS) = \{b\}$



Example 2 revisited

$$\begin{array}{l} S \rightarrow abA \mid aa \\ A \rightarrow bb \mid bS \end{array}$$
$$\begin{array}{l} \text{lookAhead}(S \rightarrow abA) = \{a\} \\ \text{lookAhead}(S \rightarrow aa) = \{a\} \\ \text{lookAhead}(A \rightarrow bb) = \{b\} \\ \text{lookAhead}(A \rightarrow bS) = \{b\} \end{array}$$

This grammar is not LL(1) (but it is LL(2)).



The need for grammar analysis

In order to determine lookahead sets systematically, we need the following information:

- ▶ 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 some derivation.



Derivation of the empty string

Definition

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

$$\text{empty } N = N \Rightarrow^* \varepsilon$$



Derivation of the empty string

Definition

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

$$\text{empty } N = N \Rightarrow^* \varepsilon$$

Example

$$\begin{array}{l} S \rightarrow AaS \mid B \\ A \rightarrow cS \mid \varepsilon \\ B \rightarrow b \end{array}$$

Here, empty A holds, but empty S and empty B do not.



First sets

Definition

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 :

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



First sets

Definition

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 :

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

Example

$$\begin{aligned} S &\rightarrow AaS \mid B \\ A &\rightarrow cS \mid \varepsilon \\ B &\rightarrow b \end{aligned}$$



First sets

Definition

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 :

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

Example

$$\begin{array}{l} S \rightarrow AaS \mid B \\ A \rightarrow cS \mid \varepsilon \\ B \rightarrow b \end{array}$$

$$\begin{array}{l} \text{first } S = \{a, b, c\} \\ \text{first } A = \{c\} \\ \text{first } B = \{b\} \end{array}$$



Follow set

Definition

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

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



Follow set

Definition

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

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

Example

$$\begin{array}{l} S \rightarrow AaS \mid B \\ A \rightarrow cS \mid \varepsilon \\ B \rightarrow b \end{array}$$



Follow set

Definition

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

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

Example

$$\begin{aligned} S &\rightarrow AaS \mid B \\ A &\rightarrow cS \mid \varepsilon \\ B &\rightarrow b \end{aligned}$$

$$\begin{aligned} \text{follow } S &= \{a\} \\ \text{follow } A &= \{a\} \\ \text{follow } B &= \{a\} \end{aligned}$$



Lookahead sets, again

The **lookahead set** production $N \rightarrow \alpha$ contains:

- ▶ Terminals which can appear as **first** of α ,
- ▶ If α can derive the **empty** string, all the terminals which may **follow** N in a derivation.

$\text{lookAhead}(A \rightarrow \beta) = \text{first } \beta \cup$
if empty β **then** follow A **else** \emptyset



Lookahead sets, again

The **lookahead set** production $N \rightarrow \alpha$ contains:

- ▶ Terminals which can appear as **first** of α ,
- ▶ If α can derive the **empty** string, all the terminals which may **follow** N in a derivation.

$$\text{lookAhead} (A \rightarrow \beta) = \text{first } \beta \cup \text{if empty } \beta \text{ then follow } A \text{ else } \emptyset$$

We need to extend the notion of first and empty to any string.



13.4 Computing empty, first, and follow



Computing empty

The following equations hold:

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

$$\text{emptyRhs } \varepsilon = \text{True}$$

$$\text{emptyRhs } (B\beta) = \text{empty } B \wedge \text{emptyRhs } \beta$$

$$\text{emptyRhs } (x\beta) = \text{False}$$



Computing empty

The following equations hold:

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

$$\text{emptyRhs } \varepsilon = \text{True}$$

$$\text{emptyRhs } (B\beta) = \text{empty } B \wedge \text{emptyRhs } \beta$$

$$\text{emptyRhs } (x\beta) = \text{False}$$

Alas, this recursive definition **cannot** be used directly as an implementation.



Computing empty

The following equations hold:

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

$$\text{emptyRhs}_n \varepsilon = \text{True}$$

$$\text{emptyRhs}_n (B\beta) = \text{empty}_{n-1} B \wedge \text{emptyRhs}_n \beta$$

$$\text{emptyRhs}_n (x\beta) = \text{False}$$

If we read it as an **iterative** algorithm, we can.



Computing empty – continued

To start, we define

$$\{ A \mid \text{empty}_0 A = \text{False} \}$$

If we look at the sequence of sets

$$\begin{aligned} & \{ A \mid \text{empty}_0 A \} \\ & \{ A \mid \text{empty}_1 A \} \\ & \dots \end{aligned}$$



Computing empty – continued

To start, we define

$$\text{empty}_0 A = \text{False}$$

If we look at the sequence of sets

$$\begin{aligned} & \{ A \mid \text{empty}_0 A \} \\ & \{ A \mid \text{empty}_1 A \} \\ & \dots \end{aligned}$$

- ▶ The first set is empty.
- ▶ Each set is at least as large as the previous.
- ▶ We reach a **fixed point** sooner or later.



Computing first

We use the same approach:

$$\text{first}_0 A = \emptyset$$

$$\text{first}_n A = \bigcup_{A \rightarrow \beta} (\text{firstRhs}_n \beta)$$

$$\text{firstRhs}_n \varepsilon = \emptyset$$

$$\text{firstRhs}_n (B\beta) = \text{first}_{n-1} B \cup \begin{cases} \text{firstRhs}_n \beta & \text{if empty } B \\ \emptyset & \text{else} \end{cases}$$

$$\text{firstRhs}_n (x\beta) = \{x\}$$

The sequence $\{(A, \text{first}_n A) \mid A \text{ nonterminal}\}$ will eventually reach a fixed point.



Computing follow

Once again the same approach:

$$\text{follow}_0 A = \emptyset$$

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

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

The sequence $\{(A, \text{follow}_n A) \mid A \text{ nonterminal}\}$ will eventually reach a fixed point.



Computing lookAhead

Using first and follow, we can finally define lookAhead:

```
lookAhead (A → β) = firstRhs β ∪  
                    if emptyRhs β then follow A  
                    else ∅
```



Computing lookAhead

Using first and follow, we can finally define lookAhead:

lookAhead ($A \rightarrow \beta$) = firstRhs $\beta \cup$
if emptyRhs β then follow A
else \emptyset

lookAhead ($A \rightarrow \beta$) = followRhs A β



Example 1, revisited again

$$\begin{array}{l} S \rightarrow cA \mid b \\ A \rightarrow cBC \mid bSA \mid a \\ B \rightarrow cc \mid Cb \\ C \rightarrow aS \mid ba \end{array}$$


Example 1, revisited again

$S \rightarrow cA \mid b$
 $A \rightarrow cBC \mid bSA \mid a$
 $B \rightarrow cc \mid Cb$
 $C \rightarrow aS \mid ba$

$\text{lookAhead}(S \rightarrow cA) = \{c\}$
 $\text{lookAhead}(S \rightarrow b) = \{b\}$
 $\text{lookAhead}(A \rightarrow cBC) = \{c\}$
 $\text{lookAhead}(A \rightarrow bSA) = \{b\}$
 $\text{lookAhead}(A \rightarrow a) = \{a\}$
 $\text{lookAhead}(B \rightarrow cc) = \{c\}$
 $\text{lookAhead}(B \rightarrow Cb) = \{a, b\}$
 $\text{lookAhead}(C \rightarrow aS) = \{a\}$
 $\text{lookAhead}(C \rightarrow ba) = \{b\}$



empty for example 1

$$\text{empty}_0 A = \text{False}$$

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

$$\text{emptyRhs}_n \varepsilon = \text{True}$$

$$\text{emptyRhs}_n (B\beta) = \text{empty}_{n-1} B \wedge \text{emptyRhs}_n \beta$$

$$\text{emptyRhs}_n (x\beta) = \text{False}$$

- ▶ 0th iteration, initialize everything to False:

$$\text{empty}_0 S = \text{False}$$

$$\text{empty}_0 A = \text{False}$$

$$\text{empty}_0 B = \text{False}$$

$$\text{empty}_0 C = \text{False}$$



empty for example 1

- ▶ 1st iteration:

$$\begin{aligned} \text{empty}_1 S &= \text{emptyRhs}_1 (cA) \vee \text{emptyRhs}_1 b \\ &= \text{False} \qquad \qquad \vee \text{False} \qquad \qquad = \text{False} \end{aligned}$$

$$\begin{aligned} \text{empty}_1 A &= \text{emptyRhs}_1 (cBC) \\ &\vee \text{emptyRhs}_1 (bSA) \\ &\vee \text{emptyRhs}_1 a \qquad \qquad \qquad = \text{False} \end{aligned}$$

$$\begin{aligned} \text{empty}_1 B &= \text{emptyRhs}_1 cc \vee \text{emptyRhs}_1 (Cb) \\ &= \text{False} \qquad \qquad \vee (\text{empty}_0 C \wedge \text{emptyRhs}_1 b) \\ &= \text{False} \qquad \qquad \vee (\text{False} \quad \wedge \text{False}) = \text{False} \end{aligned}$$

$$\text{empty}_1 C = \text{emptyRhs}_1 (aS) \vee \text{emptyRhs}_1 ba \quad = \text{False}$$

- ▶ $\text{empty}_1 = \text{empty}_0$, so we have our fixed point.



first for example 1

$$\text{first}_0 A = \emptyset$$

$$\text{first}_n A = \bigcup_{A \rightarrow \beta} (\text{firstRhs}_n \beta)$$

$$\text{firstRhs}_n \varepsilon = \emptyset$$

$$\text{firstRhs}_n (B\beta) = \text{first}_{n-1} B \cup \begin{cases} \text{if empty } B & \text{then } \text{firstRhs}_n \beta \\ & \text{else } \emptyset \end{cases}$$

$$\text{firstRhs}_n (\mathbf{x}\beta) = \{\mathbf{x}\}$$

- ▶ 0th iteration, initialize everything to \emptyset .



first for example 1

- ▶ 1st iteration:

$$\begin{aligned} \text{first}_1 S &= \text{firstRhs}_1 (cA) \cup \text{firstRhs}_1 b \\ &= \{c\} \cup \{b\} = \{c, b\} \end{aligned}$$

$$\begin{aligned} \text{first}_1 A &= \text{firstRhs}_1 (cBC) \cup \text{firstRhs}_1 (bSA) \cup \text{firstRhs}_1 a \\ &= \{c, b, a\} \end{aligned}$$

$$\begin{aligned} \text{first}_1 B &= \text{firstRhs}_1 cc \cup \text{firstRhs}_1 (Cb) \\ &= \{c\} \cup \text{first}_0 C \cup \text{if empty } C \text{ then firstRhs}_1 b \text{ else } \emptyset \\ &= \{c\} \cup \emptyset \cup \emptyset = \{c\} \end{aligned}$$

$$\text{first}_1 C = \text{firstRhs}_1 (aS) \cup \text{firstRhs}_1 ba = \{a, b\}$$



first for example 1

- ▶ 2nd iteration:

$$\text{first}_2 S = \text{firstRhs}_2 (cA) \cup \text{firstRhs}_2 b = \{b, c\}$$

$$\begin{aligned}\text{first}_2 A &= \text{firstRhs}_2 (cBC) \cup \text{firstRhs}_2 (bSA) \cup \text{firstRhs}_2 a \\ &= \{a, b, c\}\end{aligned}$$

$$\begin{aligned}\text{first}_2 B &= \text{firstRhs}_2 cc \cup \text{firstRhs}_2 (Cb) \\ &= \{c\} \cup \text{first}_2 C \cup \text{if empty } C \text{ then firstRhs}_2 b \text{ else } \emptyset \\ &= \{c\} \cup \{a, b\} \cup \emptyset = \{a, b, c\}\end{aligned}$$

$$\text{first}_2 C = \text{firstRhs}_2 (aS) \cup \text{firstRhs}_2 ba = \{a, b\}$$

- ▶ $\text{first}_3 = \text{first}_2$, so we have our fixed point.



lookAhead for example 1

lookAhead ($A \rightarrow \beta$) = firstRhs $\beta \cup$
if emptyRhs β then follow A
else \emptyset

Do we need to compute follow?



lookAhead for example 1

lookAhead ($A \rightarrow \beta$) = firstRhs $\beta \cup$
if emptyRhs β **then** follow A
else \emptyset

Do we need to compute follow?

No, because emptyRhs never returns True for this grammar.

All productions start with:

- ▶ a terminal, or
- ▶ the nonterminal C, which is not empty.



lookAhead for example 1

$$\text{lookAhead } (S \rightarrow cA) = \text{firstRhs } (cA) = \{c\}$$

$$\text{lookAhead } (S \rightarrow b) = \text{firstRhs } b = \{b\}$$

$$\text{lookAhead } (A \rightarrow cBC) = \text{firstRhs } (cBC) = \{c\}$$

$$\text{lookAhead } (A \rightarrow bSA) = \text{firstRhs } (bSA) = \{b\}$$

$$\text{lookAhead } (A \rightarrow a) = \text{firstRhs } a = \{a\}$$

$$\text{lookAhead } (B \rightarrow cc) = \text{firstRhs } cc = \{c\}$$

$$\begin{aligned} \text{lookAhead } (B \rightarrow Cb) &= \text{firstRhs } (Cb) \\ &= \text{first } C \cup \text{if empty } C \text{ then } \dots \text{ else } \emptyset \\ &= \{a, b\} \cup \emptyset = \{a, b\} \end{aligned}$$

$$\text{lookAhead } (C \rightarrow aS) = \text{firstRhs } (aS) = \{a\}$$

$$\text{lookAhead } (C \rightarrow ba) = \text{firstRhs } ba = \{b\}$$

