Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Talen en Compilers

2022 - 2023, period 2

David van Balen

Department of Information and Computing Sciences
Utrecht University

2023-01-16

13. LL parsing

13.1 Motivation

Time complexity of parsing

- Earley's algorithm and CYK 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!

This lecture

LL parsing

- Motivation
- Pushdown automata (PDA)
- PDA == context-free grammars
- LL
- Good expansion choices
- LL(1) grammars
- Computing empty, first, and follow

13.2 Pushdown automata (PDA)

We start with a reasonable question

Is there something similar to finite state automata, but for context-free languages?

We start with a reasonable question

Is there something similar to finite state automata, but for context-free languages?

Yes, **pushdown automata** (PDA)!

We start with a reasonable question

Is there something similar to finite state automata, but for context-free languages?

Yes, **pushdown automata** (PDA)!

Roughly: pushdown automaton = finite state automaton + stack.

Pushdown automata (PDA)

An extension of non-deterministic finite automata.

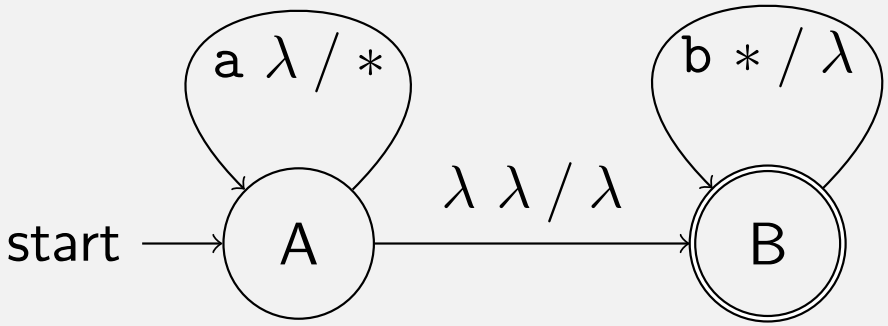
- We have an additional alphabet Γ of **stack symbols**.
- Transitions take the general form:

  $$\mathsf{d} :: \mathsf{Q} \to \mathsf{Maybe}\ \mathsf{X} \to \mathsf{Maybe}\ \Gamma \to (\mathsf{Q}, \mathsf{Maybe}\ \Gamma)$$

  where we represent absence (Nothing) with the symbol λ.
  - a A / B: if A is the top of the stack, pop it and push B,
  - a λ / B: push a B regardless of the state of the stack,
  - a A / λ: transition only if the top of the stack has an A, pop it, but do not push anything.

A word is accepted iff after reading the word, we are in an end state **and** the stack is empty.

PDA for $\{a^n b^n \mid n \in \mathbb{N}\}$



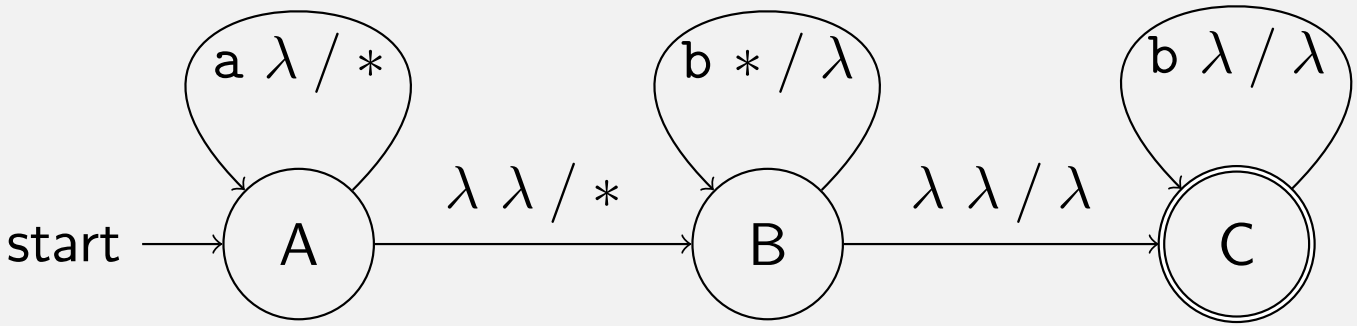
- The stack alphabet is $\{*\}$.
- We push $*$ for each a we find, and then we pop for each b.
- Remember that the word is only accepted if in the end state the stack is empty.
- Nondeterministic: a $\lambda/*$ overlaps with $\lambda\ \lambda\ /\ \lambda$.

PDA for $\{a^m b^n \mid m, n \in \mathbb{N}, m < n\}$

Try to write the PDA by modifying the previous one.

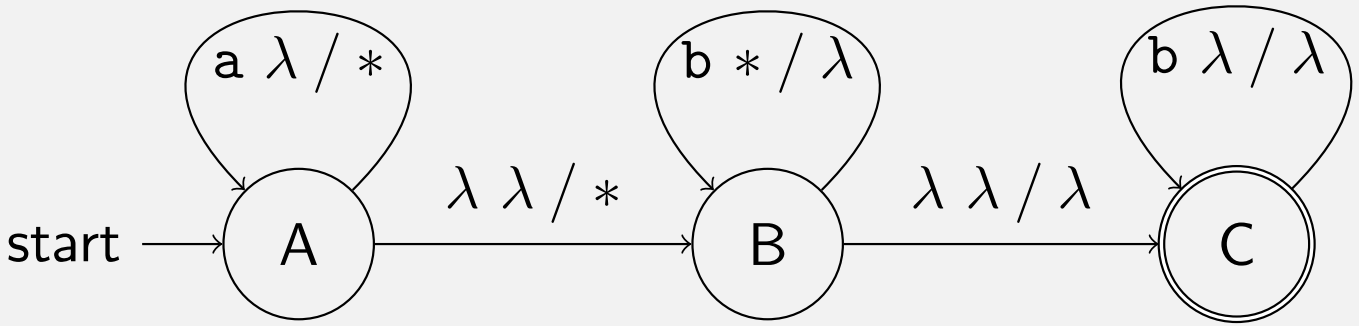
PDA for $\{a^m b^n \mid m, n \in \mathbb{N}, m < n\}$

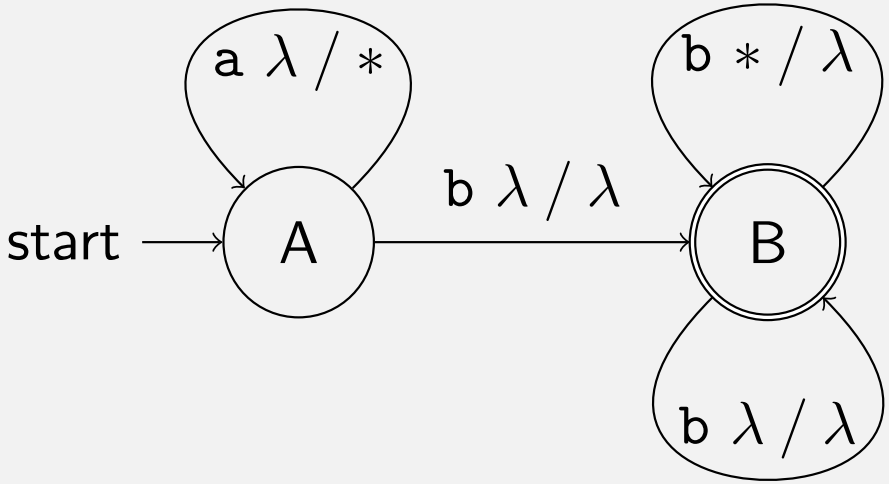
Try to write the PDA by modifying the previous one.



PDA for $\{a^m b^n \mid m, n \in \mathbb{N}, m < n\}$

Try to write the PDA by modifying the previous one.





PDA with extended transitions

Transitions may push more than one symbol onto the stack:

$$\mathsf{d} :: \mathsf{Q} \to \mathsf{Maybe}\ \mathsf{X} \to \mathsf{Maybe}\ \Gamma \to \mathsf{Q}, [\Gamma]$$

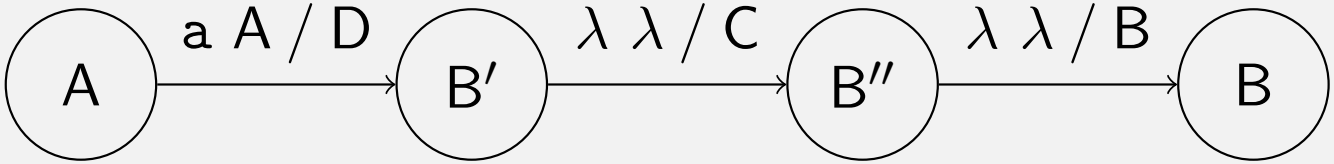
PDA with extended transitions

Transitions may push more than one symbol onto the stack:

$$\mathsf{d} :: \mathsf{Q} \to \mathsf{Maybe}\ \mathsf{X} \to \mathsf{Maybe}\ \Gamma \to \mathsf{Q}, [\Gamma]$$

We can transform such an extended PDA into a regular one:

- For each transition which pushes more than one symbol, introduce new states which consecutively push one symbol.



13.3 PDA == context-free grammars

From context-free grammars to PDA

For each context-free grammar, we can build a PDA which accepts the same language.

From context-free grammars to PDA

For each context-free grammar, we can build a PDA which accepts the same language.

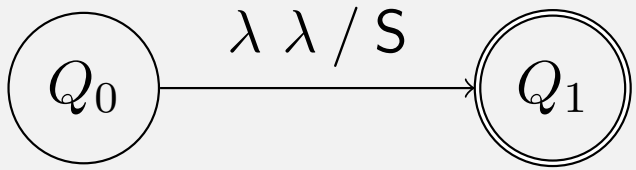
Construction

- The input alphabet consists of the terminals.
- The stack alphabet consists of the terminals and nonterminals.
- We have two states, a starting Q_0 and an accepting Q_1.

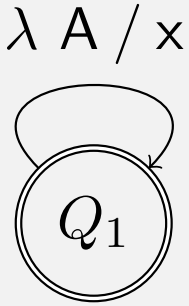
From context-free grammars to PDA

Construction – transitions

- Start the computation:



- For each production A $\rightarrow$ x:



- For each terminal x:

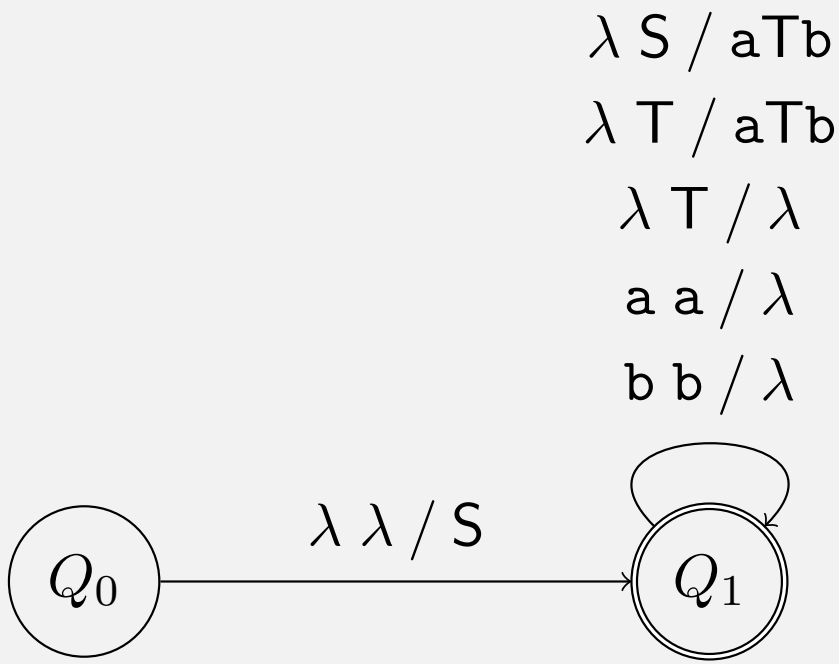
Expand-Match parsing algorithm

This algorithm is called **Expand-Match** in the Lecture Notes.

1. First we always push the start symbol S.
2. 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.
   - **Match** If the top symbol on the stack is a terminal and the first symbol of the input matches, we remove the symbol from both stack and input.
3. If the stack is empty after all the input (we are guaranteed to be in a final state), we succeed.

PDA for $\{a^n b^n \mid n \in \mathbb{N}, n > 0\}$, using Expand-Match

$$S \to aTb$$
$$T \to aTb \mid \varepsilon$$



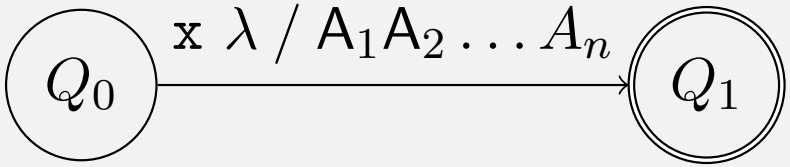
From GNF to PDA

If the grammar is in GNF, we can do a bit better.

Modified construction – transitions

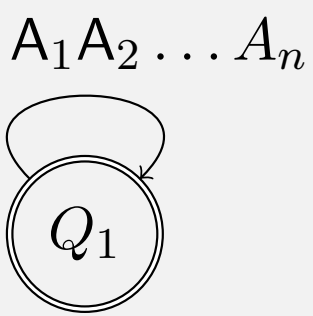
- For each rule with the start symbol on the left

  $S \to xA_1A_2 \ldots A_n$



- For each rule with another symbol on the left

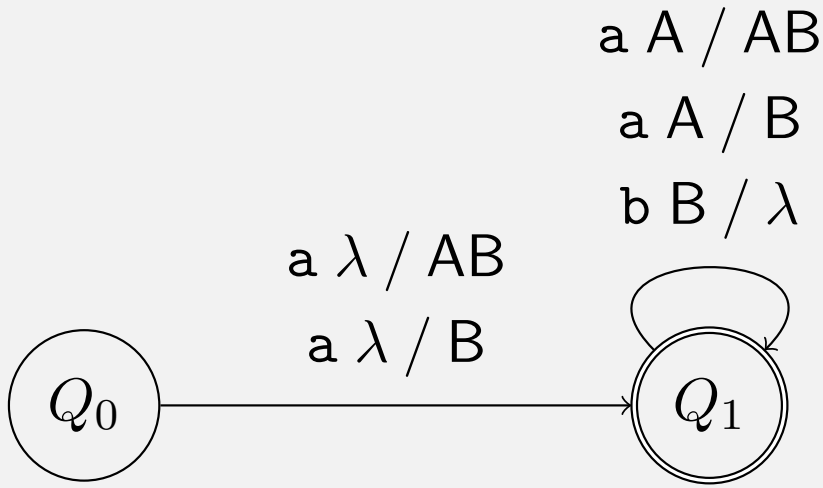
  $A \to xA_1A_2 \ldots A_n$



PDA for $\{a^n b^n \mid n \in \mathbb{N}, n > 0\}$, from a grammar

Greibach N.F.:

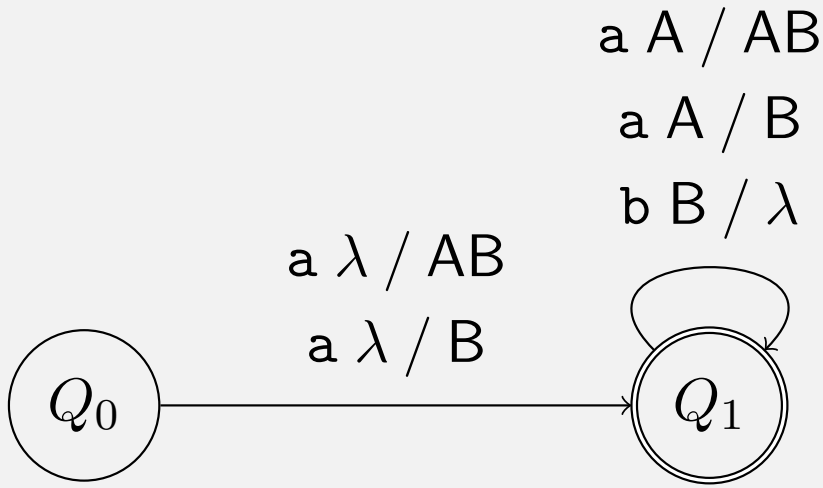
$$
\begin{aligned}
S &\to aAB \mid aB \\
A &\to aAB \mid aB \\
B &\to b
\end{aligned}
$$



PDA for $\{a^n b^n \mid n \in \mathbb{N}, n > 0\}$, from a grammar

Greibach N.F.:

$$
\begin{aligned}
S &\to aAB \mid aB \\
A &\to aAB \mid aB \\
B &\to b
\end{aligned}
$$



At every step we **consume** a character

- Less non-determinism $\Longrightarrow$ fewer options to search
- linear number of steps

Deterministic PDA

The PDA presented here are **non-deterministic**.

- Deterministic PDA have at most one choice per combination of input symbol and top of the stack.
- Linear time and linear space parsing.

Question

Are deterministic PDA as powerful as non-deterministic ones?

Deterministic PDA

The PDA presented here are **non-deterministic**.

- Deterministic PDA have at most one choice per combination of input symbol and top of the stack.
- Linear time and linear space parsing.

Question

Are deterministic PDA as powerful as non-deterministic ones?

No, they accept a **smaller** set of languages.

Indeed, deterministic PDA can only accept unambiguous context-free languages (and not even all of those).

13.4 LL

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

$S \to aS \mid cS \mid b$

Let us parse aab:

stack	input	
S	aab	initial state – expand, but which production?

Example

$S \to aS \mid cS \mid b$

Let us parse aab:

stack	input	
S	aab	
cS	aab	one option – match fails, thus error

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Example

$S \to aS \mid cS \mid b$

Let us parse aab:

stack	input	
S	aab	
aS	aab	other option – match succeeds

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Example

$S \to aS \mid cS \mid b$

Let us parse aab:

stack	input	
S	aab	
aS	aab	
S	ab	expand again

Example

$S \to aS \mid cS \mid b$

Let us parse aab:

stack	input	
S	aab	
aS	aab	
S	ab	
aS	ab	match succeeds

Example

$S \to aS \mid cS \mid b$

Let us parse aab:

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

Example

$S \to aS \mid cS \mid b$

Let us parse aab:

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

Example

$S \to aS \mid cS \mid b$

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 **l**eft,
- The machine produces a **l**eftmost derivation.

LL stack machines

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

- The machine traverses the input starting from the **l**eft,
- The machine produces a **l**eftmost 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 that leads to success (similar to NFA acceptance).

13.5 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:

$$S \to aS \mid cS \mid b$$

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:

$$S \to aS \mid cS \mid b$$

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

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

Let us parse ccccba.

Example 1

S	ccccba	
cA	ccccba	match

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

Let us parse ccccba.

Example 1

S	ccccba	
cA	ccccba	
A	cccba	expand

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

Let us parse ccccba.

Example 1

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

S	ccccba	
cA	ccccba	
A	cccba	
cBC	cccba	match

Let us parse ccccba.

Example 1

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

S	ccccba	
cA	ccccba	
A	cccba	
cBC	cccba	
BC	ccba	expand, but how?

Let us parse ccccba.

Example 1

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

S	ccccba	
cA	ccccba	
A	cccba	
cBC	cccba	
BC	ccba	expand, but how?

Let us parse ccccba.

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

Example 1

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

S	ccccba	
cA	ccccba	
A	cccba	
cBC	cccba	
BC	ccba	
ccC	ccba	match

Let us parse ccccba.

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

Example 1

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

S	ccccba	
cA	ccccba	
A	cccba	
cBC	cccba	
BC	ccba	
ccC	ccba	
cC	cba	match

Let us parse ccccba.

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

Example 1

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

Let us parse ccccba.

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

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

Example 1

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

Let us parse ccccba.

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

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

Example 1

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

Let us parse ccccba.

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

parse successful

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

Example 2

S abbb expand, but how?

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

Let us parse abbb.

Example 2

S abbb expand, but how?

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

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	match twice

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

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

$$\begin{aligned} S &\to abA \mid aa \\ A &\to bb \mid bS \end{aligned}$$

S	abbb
abA	abbb
bA	bbb
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

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

S	abbb	
abA	abbb	
bA	bbb	
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

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

Let us parse abbb.

S	abbb	
abA	abbb	
bA	bbb	
A	bb	
bb	bb	match the rest

- 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

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

Let us parse abbb.

S	abbb	
abA	abbb	
bA	bbb	
A	bb	
bb	bb	
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?

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

Let us parse `acbab`.

Example 3

S `acbab` expand, but how?

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

Let us parse `acbab`.

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

Example 3

S	acbab	
AaS	acbab	expand, but how?

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

Let us parse acbab.

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

Example 3

$$\begin{array}{rll} S & \texttt{acbab} & \\ AaS & \texttt{acbab} & \text{expand, but how?} \end{array}$$

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

Let us parse acbab.

- Because A can derive the **empty** string, the production $S \to AaS$ 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

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

S	acbab	
AaS	acbab	
aS	acbab	match

Let us parse acbab.

- Because A can derive the **empty** string, the production $S \to AaS$ 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

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

S	acbab	
AaS	acbab	
aS	acbab	
S	cbab	expand

Let us parse acbab.

- Because A can derive the **empty** string, the production $S \to AaS$ 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

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

S	acbab	
AaS	acbab	
aS	acbab	
S	cbab	
AaS	cbab	expand

Let us parse acbab.

- Because A can derive the **empty** string, the production $S \to AaS$ 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

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

Let us parse acbab.

S	acbab	
AaS	acbab	
aS	acbab	
S	cbab	
AaS	cbab	
cSaS	cbab	match

- Because A can derive the **empty** string, the production $S \to AaS$ 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

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

Let us parse acbab.

S	acbab	
AaS	acbab	
aS	acbab	
S	cbab	
AaS	cbab	
cSaS	cbab	
SaS	bab	rest is easy

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

13.6 LL(1) grammars

Definition of LL(1) grammars

Definition

The **lookahead set** of a production $\mathsf{N} \to \alpha$ is defined as

$$\mathsf{lookAhead}\ (\mathsf{N} \to \alpha) = \{\, \mathtt{x} \mid \mathsf{S} \Rightarrow^* \gamma \mathsf{N} \delta \Rightarrow \gamma\alpha\delta \Rightarrow^* \gamma \mathtt{x} \beta \,\}$$

Definition of LL(1) grammars

Definition

The **lookahead set** of a production $\mathsf{N} \to \alpha$ is defined as

$$\mathsf{lookAhead}\ (\mathsf{N} \to \alpha) = \{\, \mathtt{x} \mid \mathsf{S} \Rightarrow^* \gamma \mathsf{N} \delta \Rightarrow \gamma\alpha\delta \Rightarrow^* \gamma \mathtt{x} \beta \,\}$$

Definition

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

$$\mathsf{lookAhead}\ (\mathsf{N} \to \alpha) \cap \mathsf{lookAhead}\ (\mathsf{N} \to \beta) = \emptyset$$

for all pairs of different productions $\mathsf{N} \to \alpha$, $\mathsf{N} \to \beta$.

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:

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

Example 1 revisited

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

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

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

Example 1 revisited

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

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

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

Example 1 revisited

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

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

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

Example 1 revisited

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

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

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

This grammar is LL(1).

Example 2 revisited

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

Example 2 revisited

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

$$
\begin{array}{l}
\mathsf{lookAhead}\ (S \to abA) = \\
\mathsf{lookAhead}\ (S \to aa) \ = \\
\mathsf{lookAhead}\ (A \to bb) \ = \\
\mathsf{lookAhead}\ (A \to bS) \ =
\end{array}
$$

Example 2 revisited

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

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

Example 2 revisited

$$\begin{array}{l} \mathsf{S} \to \mathtt{abA} \mid \mathtt{aa} \\ \mathsf{A} \to \mathtt{bb} \mid \mathtt{bS} \end{array}$$

$$\begin{array}{l} \mathsf{lookAhead}\ (\mathsf{S} \to \mathtt{abA}) = \{\mathtt{a}\} \\ \mathsf{lookAhead}\ (\mathsf{S} \to \mathtt{aa}) = \{\mathtt{a}\} \\ \mathsf{lookAhead}\ (\mathsf{A} \to \mathtt{bb}) = \{\mathtt{b}\} \\ \mathsf{lookAhead}\ (\mathsf{A} \to \mathtt{bS}) = \{\mathtt{b}\} \end{array}$$

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

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

Example

> $S \to AaS \mid B$
> $A \to cS \mid \varepsilon$
> $B \to b$

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:

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

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:

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

Example

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

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

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

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

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:

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

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:

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

Example

$$\begin{aligned} \mathsf{S} &\to \mathsf{AaS} \mid \mathsf{B} \\ \mathsf{A} &\to \mathtt{c}\mathsf{S} \mid \varepsilon \\ \mathsf{B} &\to \mathtt{b} \end{aligned}$$

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:

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

Example

$$\begin{aligned} \text{S} &\to \text{AaS} \mid \text{B} \\ \text{A} &\to \mathtt{c}\text{S} \mid \varepsilon \\ \text{B} &\to \mathtt{b} \end{aligned}$$

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

Lookahead sets, again

The **lookahead set** production $N \to \alpha$ 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.

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

Lookahead sets, again

The **lookahead set** production $N \to \alpha$ 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.

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

We need to extend the notion of first and empty to strings.

13.7 Computing empty, first, and follow

Computing empty

The following equations hold:

$$
\begin{aligned}
&\mathsf{empty}\ \ \mathsf{A} = \bigvee_{\mathsf{A}\to\beta}(\mathsf{emptyRhs}\ \ \beta)\\
&\mathsf{emptyRhs}\ \ \varepsilon\ \ \ \ \ \ = \mathsf{True}\\
&\mathsf{emptyRhs}\ \ (\mathsf{B}\beta) = \mathsf{empty}\ \ \ \ \ \ \mathsf{B} \wedge \mathsf{emptyRhs}\ \ \beta\\
&\mathsf{emptyRhs}\ \ (\mathtt{x}\beta) = \mathsf{False}
\end{aligned}
$$

Computing empty

The following equations hold:

$$
\begin{aligned}
&\mathsf{empty}\ \ \mathsf{A} = \bigvee_{\mathsf{A}\to\beta}(\mathsf{emptyRhs}\ \ \beta)\\
&\mathsf{emptyRhs}\ \ \varepsilon\ \ \ \ \ \ = \mathsf{True}\\
&\mathsf{emptyRhs}\ \ (\mathsf{B}\beta) = \mathsf{empty}\ \ \ \ \ \ \mathsf{B} \wedge \mathsf{emptyRhs}\ \ \beta\\
&\mathsf{emptyRhs}\ \ (\mathtt{x}\beta) = \mathsf{False}
\end{aligned}
$$

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

Computing empty

The following equations hold:

$$
\begin{aligned}
&\mathsf{empty_n}\ \mathsf{A} = \bigvee_{\mathsf{A}\to\beta} (\mathsf{emptyRhs_n}\ \beta) \\
&\mathsf{emptyRhs_n}\ \varepsilon \quad\ = \mathsf{True} \\
&\mathsf{emptyRhs_n}\ (\mathsf{B}\beta) = \mathsf{empty_{n-1}}\ \mathsf{B} \wedge \mathsf{emptyRhs_n}\ \beta \\
&\mathsf{emptyRhs_n}\ (\mathtt{x}\beta) = \mathsf{False}
\end{aligned}
$$

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

Computing empty – continued

To start, we define

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

If we look at the sequence of sets

$$\{\mathsf{A} \mid \text{empty}_0\ \mathsf{A}\}$$
$$\{\mathsf{A} \mid \text{empty}_1\ \mathsf{A}\}$$
$$\ldots$$

Computing empty – continued

To start, we define

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

If we look at the sequence of sets

$$\{A \mid \text{empty}_0\ A\}$$
$$\{A \mid \text{empty}_1\ A\}$$
$$\ldots$$

- 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:

$$
\begin{aligned}
&\mathsf{first}_0\ \mathsf{A} = \emptyset \\
&\mathsf{first}_n\ \mathsf{A} = \bigcup_{\mathsf{A}\to\beta} (\mathsf{firstRhs}_n\ \beta) \\
&\mathsf{firstRhs}_n\ \varepsilon \qquad = \emptyset \\
&\mathsf{firstRhs}_n\ (\mathsf{B}\beta) = \mathsf{first}_{n-1}\ \mathsf{B} \cup \textbf{if}\ \mathsf{empty}\ \mathsf{B}\ \textbf{then}\ \mathsf{firstRhs}_n\ \beta \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else}\ \ \emptyset \\
&\mathsf{firstRhs}_n\ (\mathtt{x}\beta) = \{\mathtt{x}\}
\end{aligned}
$$

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

Computing follow

Once again the same approach:

$$\begin{aligned}
&\mathsf{follow}_0\ \mathsf{A} = \emptyset \\
&\mathsf{follow}_n\ \mathsf{A} = \bigcup_{\mathsf{B}\to\alpha\mathsf{A}\beta}(\mathsf{followRhs}_n\ \mathsf{B}\ \beta) \\
&\mathsf{followRhs}_n\ \mathsf{B}\ \beta = \mathsf{firstRhs}\ \beta \cup \textbf{if}\ \mathsf{emptyRhs}\ \beta\ \textbf{then}\ \mathsf{follow}_{n-1}\ \mathsf{B}\ \textbf{else}\ \emptyset
\end{aligned}$$

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

Computing lookAhead

Using first and follow, we can finally define lookAhead:

$$\begin{aligned} \text{lookAhead } (A \to \beta) = \text{firstRhs } \beta \cup \\ \textbf{if } \text{emptyRhs } \beta \textbf{ then } \text{follow } A \\ \textbf{else } \emptyset \end{aligned}$$

Computing lookAhead

Using first and follow, we can finally define lookAhead:

$$\text{lookAhead } (A \to \beta) = \text{firstRhs } \beta \cup \textbf{if } \text{emptyRhs } \beta \textbf{ then } \text{follow } A \textbf{ else } \emptyset$$

$$\text{lookAhead } (A \to \beta) = \text{followRhs } A\ \beta$$

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Example 1, revisited again

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

Example 1, revisited again

$$
\begin{aligned}
S &\to cA \mid b \\
A &\to cBC \mid bSA \mid a \\
B &\to cc \mid Cb \\
C &\to aS \mid ba
\end{aligned}
$$

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

empty for example 1

$$
\begin{aligned}
&\mathsf{empty}_0\ \mathsf{A} = \mathsf{False} \\
&\mathsf{empty}_\mathsf{n}\ \mathsf{A} = \bigcup_{\mathsf{A}\to\beta}(\mathsf{emptyRhs}_\mathsf{n}\ \beta) \\
&\mathsf{emptyRhs}_\mathsf{n}\ \varepsilon \quad\ = \mathsf{True} \\
&\mathsf{emptyRhs}_\mathsf{n}\ (\mathsf{B}\beta) = \mathsf{empty}_{\mathsf{n}-1}\ \mathsf{B} \wedge \mathsf{emptyRhs}_\mathsf{n}\ \beta \\
&\mathsf{emptyRhs}_\mathsf{n}\ (\mathtt{x}\beta) = \mathsf{False}
\end{aligned}
$$

- 0th iteration, initialize everything to False:

$$
\begin{aligned}
&\mathsf{empty}_0\ \mathsf{S} = \mathsf{False} \\
&\mathsf{empty}_0\ \mathsf{A} = \mathsf{False} \\
&\mathsf{empty}_0\ \mathsf{B} = \mathsf{False} \\
&\mathsf{empty}_0\ \mathsf{C} = \mathsf{False}
\end{aligned}
$$

empty for example 1

- 1st iteration:

$$
\begin{aligned}
\mathsf{empty}_1\ \mathsf{S} &= \mathsf{emptyRhs}_1\ (\mathtt{cA}) \vee \mathsf{emptyRhs}_1\ \mathtt{b} \\
&= \mathsf{False} \qquad\qquad\ \vee \mathsf{False} \qquad\qquad = \mathsf{False} \\
\mathsf{empty}_1\ \mathsf{A} &= \mathsf{emptyRhs}_1\ (\mathtt{cBC}) \\
&\quad \vee \mathsf{emptyRhs}_1\ (\mathtt{bSA}) \\
&\quad \vee \mathsf{emptyRhs}_1\ \mathtt{a} \qquad\qquad = \mathsf{False} \\
\mathsf{empty}_1\ \mathsf{B} &= \mathsf{emptyRhs}_1\ \mathtt{cc} \vee \mathsf{emptyRhs}_1\ (\mathtt{Cb}) \\
&= \mathsf{False} \qquad\quad \vee (\mathsf{empty}_0\ \mathsf{C} \wedge \mathsf{emptyRhs}_1\ \mathtt{b}) \\
&= \mathsf{False} \qquad\quad \vee (\mathsf{False} \quad\ \wedge \mathsf{False}) = \mathsf{False} \\
\mathsf{empty}_1\ \mathsf{C} &= \mathsf{emptyRhs}_1\ (\mathtt{aS}) \vee \mathsf{emptyRhs}_1\ \mathtt{ba} \quad = \mathsf{False}
\end{aligned}
$$

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

first for example 1

$$
\begin{aligned}
&\mathsf{first}_0\ \mathsf{A} = \emptyset \\
&\mathsf{first}_\mathsf{n}\ \mathsf{A} = \bigcup_{\mathsf{A}\to\beta}(\mathsf{firstRhs}_\mathsf{n}\ \beta) \\
&\mathsf{firstRhs}_\mathsf{n}\ \varepsilon \qquad = \emptyset \\
&\mathsf{firstRhs}_\mathsf{n}\ (\mathsf{B}\beta) = \mathsf{first}_{\mathsf{n}-1}\ \mathsf{B} \cup \textbf{if}\ \mathsf{empty\ B}\ \textbf{then}\ \mathsf{firstRhs}_\mathsf{n}\ \beta \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else}\ \ \emptyset \\
&\mathsf{firstRhs}_\mathsf{n}\ (\mathtt{x}\beta) = \{\mathtt{x}\}
\end{aligned}
$$

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

first for example 1

- 1st iteration:

$$
\begin{aligned}
\mathsf{first}_1\ \mathsf{S} &= \mathsf{firstRhs}_1\ (\mathtt{cA}) \cup \mathsf{firstRhs}_1\ \mathtt{b} \\
&= \{\mathtt{c}\} \cup \{\mathtt{b}\} = \{\mathtt{c}, \mathtt{b}\} \\
\mathsf{first}_1\ \mathsf{A} &= \mathsf{firstRhs}_1\ (\mathtt{cBC}) \cup \mathsf{firstRhs}_1\ (\mathtt{bSA}) \cup \mathsf{firstRhs}_1\ \mathtt{a} \\
&= \{\mathtt{c}, \mathtt{b}, \mathtt{a}\} \\
\mathsf{first}_1\ \mathsf{B} &= \mathsf{firstRhs}_1\ \mathtt{cc} \cup \mathsf{firstRhs}_1\ (\mathsf{Cb}) \\
&= \{\mathtt{c}\} \cup \mathsf{first}_0\ \mathsf{C} \cup \textbf{if}\ \mathsf{empty}\ \mathsf{C}\ \textbf{then}\ \mathsf{firstRhs}_1\ \mathtt{b}\ \textbf{else}\ \emptyset \\
&= \{\mathtt{c}\} \cup \emptyset \cup \emptyset = \{\mathtt{c}\} \\
\mathsf{first}_1\ \mathsf{C} &= \mathsf{firstRhs}_1\ (\mathtt{aS}) \cup \mathsf{firstRhs}_1\ \mathtt{ba} = \{\mathtt{a}, \mathtt{b}\}
\end{aligned}
$$

first for example 1

- 2nd iteration:

$$
\begin{aligned}
\mathrm{first}_2\ \mathrm{S} &= \mathrm{firstRhs}_2\ (\mathtt{cA}) \cup \mathrm{firstRhs}_2\ \mathtt{b} &&= \{\mathtt{b}, \mathtt{c}\} \\
\mathrm{first}_2\ \mathrm{A} &= \mathrm{firstRhs}_2\ (\mathtt{cBC}) \cup \mathrm{firstRhs}_2\ (\mathtt{bSA}) \cup \mathrm{firstRhs}_2\ \mathtt{a} \\
&&&= \{\mathtt{a}, \mathtt{b}, \mathtt{c}\} \\
\mathrm{first}_2\ \mathrm{B} &= \mathrm{firstRhs}_2\ \mathtt{cc} \cup \mathrm{firstRhs}_2\ (\mathtt{Cb}) \\
&= \{\mathtt{c}\} \cup \mathrm{first}_2\ \mathrm{C} \cup \textbf{if}\ \mathrm{empty}\ \mathrm{C}\ \textbf{then}\ \mathrm{firstRhs}_2\ \mathtt{b}\ \textbf{else}\ \emptyset \\
&= \{\mathtt{c}\} \cup \{\mathtt{a}, \mathtt{b}\} \cup \emptyset &&= \{\mathtt{a}, \mathtt{b}, \mathtt{c}\} \\
\mathrm{first}_2\ \mathrm{C} &= \mathrm{firstRhs}_2\ (\mathtt{aS}) \cup \mathrm{firstRhs}_2\ \mathtt{ba} &&= \{\mathtt{a}, \mathtt{b}\}
\end{aligned}
$$

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

lookAhead **for example 1**

$$\text{lookAhead } (A \to \beta) = \text{firstRhs } \beta \cup \textbf{if } \text{emptyRhs } \beta \textbf{ then } \text{follow } A \textbf{ else } \emptyset$$

Do we need to compute follow?

lookAhead **for example 1**

$$\text{lookAhead } (A \to \beta) = \text{firstRhs } \beta \cup \textbf{if } \text{emptyRhs } \beta \textbf{ then } \text{follow } A \textbf{ else } \emptyset$$

Do we need to compute follow?

No, because emptyRhs never returns True for this grammar.

lookAhead **for example 1**

$$
\begin{array}{lll}
\mathsf{lookAhead}\ (\mathsf{S} \to \mathtt{cA}) & = \mathsf{firstRhs}\ (\mathtt{cA}) & = \{\mathtt{c}\} \\
\mathsf{lookAhead}\ (\mathsf{S} \to \mathtt{b}) & = \mathsf{firstRhs}\ \mathtt{b} & = \{\mathtt{b}\} \\
\mathsf{lookAhead}\ (\mathsf{A} \to \mathtt{cBC}) & = \mathsf{firstRhs}\ (\mathtt{cBC}) & = \{\mathtt{c}\} \\
\mathsf{lookAhead}\ (\mathsf{A} \to \mathtt{bSA}) & = \mathsf{firstRhs}\ (\mathtt{bSA}) & = \{\mathtt{b}\} \\
\mathsf{lookAhead}\ (\mathsf{A} \to \mathtt{a}) & = \mathsf{firstRhs}\ \mathtt{a} & = \{\mathtt{a}\} \\
\mathsf{lookAhead}\ (\mathsf{B} \to \mathtt{cc}) & = \mathsf{firstRhs}\ \mathtt{cc} & = \{\mathtt{c}\} \\
\mathsf{lookAhead}\ (\mathsf{B} \to \mathtt{Cb}) & = \mathsf{firstRhs}\ (\mathtt{Cb}) & \\
 & = \mathsf{first}\ \mathsf{C} \cup \textbf{if}\ \mathsf{empty}\ \mathsf{C}\ \textbf{then} \ldots \textbf{else}\ \emptyset & \\
 & = \{\mathtt{a}, \mathtt{b}\} \cup \emptyset & = \{\mathtt{a}, \mathtt{b}\} \\
\mathsf{lookAhead}\ (\mathsf{C} \to \mathtt{aS}) & = \mathsf{firstRhs}\ (\mathtt{aS}) & = \{\mathtt{a}\} \\
\mathsf{lookAhead}\ (\mathsf{C} \to \mathtt{ba}) & = \mathsf{firstRhs}\ \mathtt{ba} & = \{\mathtt{b}\}
\end{array}
$$