

Languages and Machines (Module 7 TCS+IAM)

L&M 4: Parallel Composition of Automata &
Grammars

Extra Material Blackboard + Ch 3:1-5

Alexander Skopalik

Discrete Mathematics and Mathematical Programming, Applied Mathematics, University
of Twente

Lecture 4

Contents

- 1 Composition of Automata in Software Engineering
 - Product of Automata
 - Parallel composition
 - Projection and Hiding
- 2 Grammars and Derivations
 - Derivations
 - Derivation trees
 - Generated Language
- 3 Regular grammars
- 4 Syntax of programming languages
 - Ambiguity
 - Naive Parsing

Contents

- 1 Composition of Automata in Software Engineering
 - Product of Automata
 - Parallel composition
 - Projection and Hiding
- 2 Grammars and Derivations
 - Derivations
 - Derivation trees
 - Generated Language
- 3 Regular grammars
- 4 Syntax of programming languages
 - Ambiguity
 - Naive Parsing

Real-Life Applications

Applications of regular languages:

- 1 Searching in text (**pattern matching**, cf. C 2.4)
- 2 **Lexical analysis** of software programs (keywords, identifiers)
- 3 **Input validation** for software security
- 4 **System specification** in software engineering

Real-Life Applications

Applications of regular languages:

- 1 Searching in text (**pattern matching**, cf. C 2.4)
- 2 **Lexical analysis** of software programs (keywords, identifiers)
- 3 **Input validation** for software security
- 4 **System specification** in software engineering

Typical approach:

- 1 Identify a number of patterns
- 2 Specify the language by a regular expression

Real-Life Applications

Applications of regular languages:

- 1 Searching in text (**pattern matching**, cf. C 2.4)
- 2 **Lexical analysis** of software programs (keywords, identifiers)
- 3 **Input validation** for software security
- 4 **System specification** in software engineering

Typical approach:

- 1 Identify a number of patterns
- 2 Specify the language by a regular expression
- 3 Transform the expression to an NFA- λ

Real-Life Applications

Applications of regular languages:

- 1 Searching in text (**pattern matching**, cf. C 2.4)
- 2 **Lexical analysis** of software programs (keywords, identifiers)
- 3 **Input validation** for software security
- 4 **System specification** in software engineering

Typical approach:

- 1 Identify a number of patterns
- 2 Specify the language by a regular expression
- 3 Transform the expression to an NFA- λ
- 4 Transform the NFA- λ into a DFA, using the subset construction

Real-Life Applications

Applications of regular languages:

- 1 Searching in text ([pattern matching](#), cf. C 2.4)
- 2 [Lexical analysis](#) of software programs (keywords, identifiers)
- 3 [Input validation](#) for software security
- 4 [System specification](#) in software engineering

Typical approach:

- 1 Identify a number of patterns
- 2 Specify the language by a regular expression
- 3 Transform the expression to an NFA- λ
- 4 Transform the NFA- λ into a DFA, using the subset construction
- 5 Minimise the DFA

Real-Life Applications

Applications of regular languages:

- 1 Searching in text ([pattern matching](#), cf. C 2.4)
- 2 [Lexical analysis](#) of software programs (keywords, identifiers)
- 3 [Input validation](#) for software security
- 4 [System specification](#) in software engineering

Typical approach:

- 1 Identify a number of patterns
- 2 Specify the language by a regular expression
- 3 Transform the expression to an NFA- λ
- 4 Transform the NFA- λ into a DFA, using the subset construction
- 5 Minimise the DFA
- 6 Translate the minimal DFA directly into code (C, Java, Python)

A regular expression for E-mail addresses

according to standard RFC 5322, cf. <http://emailregex.com>

```
(?:[a-z0-9!#$%&'*/=?^_`{|}~--]+(?:\.(?:[a-z0-9!#$%&'*/=?^_`{|}~--]+)*"?(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f]))*)@((?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)|(?:(?:25[0-5]|2[0-4][0-9]|01?[0-9])[0-9]{3}(?:25[0-5]|2[0-4][0-9]|01?[0-9])[0-9]?|[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+\.))
```

Note: this regexp uses extended notation, which you find in many practical examples, like ? for optional, [a-z] for ranges of symbols.

Application in Software Engineering

Automata are used in Software Engineering:

- **Specification** of properties and models of systems
- **Verification** of properties for models

Application in Software Engineering

Automata are used in Software Engineering:

- **Specification** of properties and models of systems
- **Verification** of properties for models

example: $\mathcal{L}(\text{efficient elevator}) \subseteq \mathcal{L}(\text{safe elevator})$

Application in Software Engineering

Automata are used in Software Engineering:

- **Specification** of properties and models of systems
- **Verification** of properties for models

example: $\mathcal{L}(\text{efficient elevator}) \subseteq \mathcal{L}(\text{safe elevator})$

Prerequisites for methods in Software Engineering:

- **Divide-and-conquer**: not monolithic, but compositional
 - **product** and **parallel composition** of languages and automata

Application in Software Engineering

Automata are used in Software Engineering:

- **Specification** of properties and models of systems
- **Verification** of properties for models

example: $\mathcal{L}(\text{efficient elevator}) \subseteq \mathcal{L}(\text{safe elevator})$

Prerequisites for methods in Software Engineering:

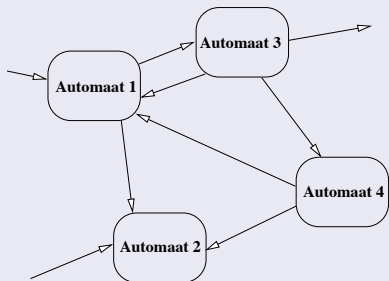
- **Divide-and-conquer**: not monolithic, but compositional
 - **product** and **parallel composition** of languages and automata
- **Information-hiding**: irrelevant details/internal behavior must be able to be suppressed (hidden)
 - **Projection** and **hiding** of languages and automata

Composition of Automata

- A system often consists of a number of components
- Components mutually exchange messages

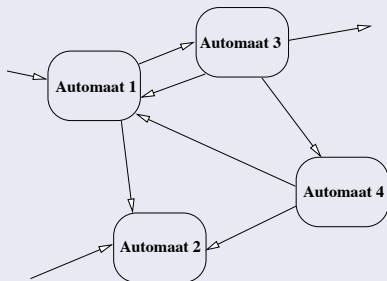
Composition of Automata

- A system often consists of a number of components
- Components mutually exchange messages
- Now we want to compose automata (DFA/NFA)



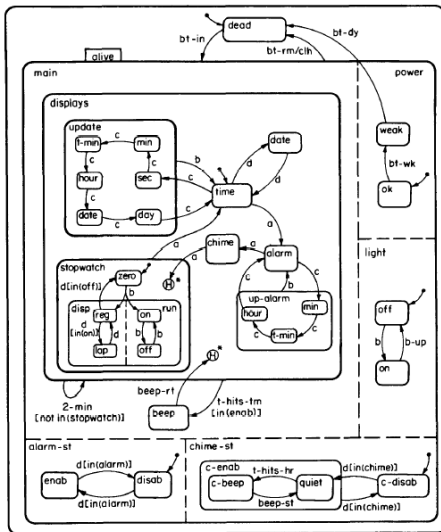
Composition of Automata

- A system often consists of a number of components
- Components mutually exchange messages
- Now we want to compose automata (DFA/NFA)



- Communication = simultaneously reading the same symbol

Remember: Smart Watch by David Harel



Using parallel components, but also hierarchical states

Product of automata

$M_1 \times M_2$: Product automata of automata M_1 and M_2

- We want $M_1 \times M_2$ to accept a word, if it is simultaneously accepted by both M_1 and M_2 .
- So we require: $\mathcal{L}(M_1 \times M_2) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$

Product of automata

$M_1 \times M_2$: Product automata of automata M_1 and M_2

- We want $M_1 \times M_2$ to accept a word, if it is simultaneously accepted by both M_1 and M_2 .
- So we require: $\mathcal{L}(M_1 \times M_2) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$

States and transitions

- The states of the automata $M_1 \times M_2$ will consist of **pairs of states** of M_1 and M_2 .
- The transitions of $M_1 \times M_2$ will consist of the **simultaneous** transitions of M_1 and M_2 .

Product of automata

$M_1 \times M_2$: Product automata of automata M_1 and M_2

- We want $M_1 \times M_2$ to accept a word, if it is simultaneously accepted by both M_1 and M_2 .
- So we require: $\mathcal{L}(M_1 \times M_2) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$

States and transitions

- The states of the automata $M_1 \times M_2$ will consist of **pairs of states** of M_1 and M_2 .
- The transitions of $M_1 \times M_2$ will consist of the **simultaneous** transitions of M_1 and M_2 .

Cartesian product:

- $P \times Q = \{(p, q) \mid p \in P \wedge q \in Q\}$
- E.g.: $\{a, b, c\} \times \{1, 2\} = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$

Product automata for DFA's (mathematical)

- Let the following (possibly incomplete) DFA's be given:

$$M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1) \text{ and } M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$$

Product automata for DFA's (mathematical)

- Let the following (possibly incomplete) DFA's be given:

$$M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1) \text{ and } M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$$

- We now define the product automata (another DFA):

$$M_1 \times M_2 := (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$$

where $\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$

Product automata for DFA's (mathematical)

- Let the following (possibly incomplete) DFA's be given:

$$M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1) \text{ and } M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$$

- We now define the product automata (another DFA):

$$M_1 \times M_2 := (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$$

where $\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$

- If q_1 or q_2 can't do a a , then (q_1, q_2) can't either
- Example:** even number of a's, odd number of b's:

Product automata for DFA's (mathematical)

- Let the following (possibly incomplete) DFA's be given:

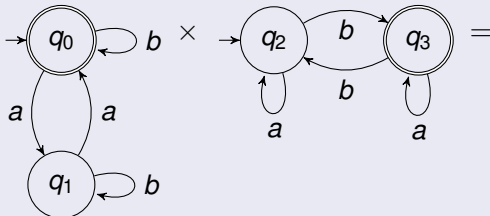
$$M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1) \text{ and } M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$$

- We now define the product automata (another DFA):

$$M_1 \times M_2 := (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$$

where $\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$

- If q_1 or q_2 can't do a a , then (q_1, q_2) can't either
- Example:** even number of a's, odd number of b's:



Product automata for DFA's (mathematical)

- Let the following (possibly incomplete) DFA's be given:

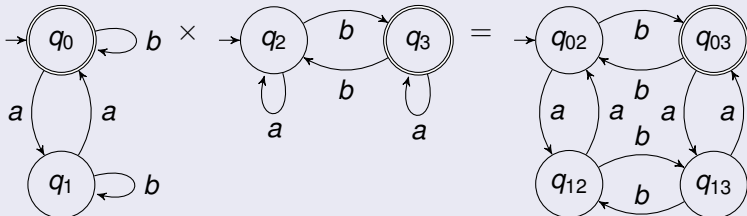
$$M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1) \text{ and } M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$$

- We now define the product automata (another DFA):

$$M_1 \times M_2 := (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$$

where $\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$

- If q_1 or q_2 can't do a a , then (q_1, q_2) can't either
- Example:** even number of a's, odd number of b's:



Product automata for NFA- λ

For NFA- λ it's almost the same:

- δ is now a relation, not a function
- We require **no** synchronisation of λ -steps
- These λ -steps thus occur independently (=interleaving).

Product automata for NFA- λ

For NFA- λ it's almost the same:

- δ is now a relation, not a function
- We require **no** synchronisation of λ -steps
- These λ -steps thus occur independently (=interleaving).

Product of two NFA- λ 's (formally)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$.

Product automata for NFA- λ

For NFA- λ it's almost the same:

- δ is now a relation, not a function
- We require **no** synchronisation of λ -steps
- These λ -steps thus occur independently (=interleaving).

Product of two NFA- λ 's (formally)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$.
- Define $M_1 \times M_2 := (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$

Product automata for NFA- λ

For NFA- λ it's almost the same:

- δ is now a relation, not a function
- We require **no** synchronisation of λ -steps
- These λ -steps thus occur independently (=interleaving).

Product of two NFA- λ 's (formally)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$.
- Define $M_1 \times M_2 := (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$
- where for $a \neq \lambda$:
 - $(q'_1, q'_2) \in \delta((q_1, q_2), a)$ if $q'_1 \in \delta_1(q_1, a)$ and $q'_2 \in \delta_2(q_2, a)$

Product automata for NFA- λ

For NFA- λ it's almost the same:

- δ is now a relation, not a function
- We require **no** synchronisation of λ -steps
- These λ -steps thus occur independently (=interleaving).

Product of two NFA- λ 's (formally)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$.
- Define $M_1 \times M_2 := (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$
- where for $a \neq \lambda$:
 - $(q'_1, q'_2) \in \delta((q_1, q_2), a)$ if $q'_1 \in \delta_1(q_1, a)$ and $q'_2 \in \delta_2(q_2, a)$
- but for $a = \lambda$:
 - $(q'_1, q'_2) \in \delta((q_1, q_2), \lambda)$ if $\begin{cases} q'_1 \in \delta_1(q_1, \lambda) \text{ and } q'_2 = q_2; \text{ or} \\ q'_2 \in \delta_2(q_2, \lambda) \text{ and } q'_1 = q_1. \end{cases}$

Better Notation

Original notation:

- $(q'_1, q'_2) \in \delta((q_1, q_2), a)$ if $q'_1 \in \delta_1(q_1, a)$ and $q'_2 \in \delta_2(q_2, a)$
- $(q'_1, q'_2) \in \delta((q_1, q_2), \lambda)$ if $\begin{cases} q'_1 \in \delta_1(q_1, a) \text{ and } q'_2 = q_2; \text{ or} \\ q'_2 \in \delta_2(q_2, a) \text{ and } q'_1 = q_1. \end{cases}$

Better Notation

Original notation:

- $(q'_1, q'_2) \in \delta((q_1, q_2), a)$ if $q'_1 \in \delta_1(q_1, a)$ and $q'_2 \in \delta_2(q_2, a)$
- $(q'_1, q'_2) \in \delta((q_1, q_2), \lambda)$ if $\begin{cases} q'_1 \in \delta_1(q_1, a) \text{ and } q'_2 = q_2; \text{ or} \\ q'_2 \in \delta_2(q_2, a) \text{ and } q'_1 = q_1. \end{cases}$

Better notation: $p \xrightarrow{a}_1 q$ for $q \in \delta_1(p, a)$

$$\frac{q_1 \xrightarrow{a}_1 q'_1 \quad q_2 \xrightarrow{a}_2 q'_2}{(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)}$$

Better Notation

Original notation:

- $(q'_1, q'_2) \in \delta((q_1, q_2), a)$ if $q'_1 \in \delta_1(q_1, a)$ and $q'_2 \in \delta_2(q_2, a)$
- $(q'_1, q'_2) \in \delta((q_1, q_2), \lambda)$ if $\begin{cases} q'_1 \in \delta_1(q_1, a) \text{ and } q'_2 = q_2; \text{ or} \\ q'_2 \in \delta_2(q_2, a) \text{ and } q'_1 = q_1. \end{cases}$

Better notation: $p \xrightarrow{a}_1 q$ for $q \in \delta_1(p, a)$

$$q_1 \xrightarrow{a}_1 q'_1 \quad q_2 \xrightarrow{a}_2 q'_2$$

$$\frac{q_1 \xrightarrow{a}_1 q'_1 \quad q_2 \xrightarrow{a}_2 q'_2}{(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)}$$

$$q_1 \xrightarrow{\lambda}_1 q'_1$$

$$\frac{q_1 \xrightarrow{\lambda}_1 q'_1}{(q_1, q_2) \xrightarrow{\lambda} (q'_1, q_2)}$$

Better Notation

Original notation:

- $(q'_1, q'_2) \in \delta((q_1, q_2), a)$ if $q'_1 \in \delta_1(q_1, a)$ and $q'_2 \in \delta_2(q_2, a)$
- $(q'_1, q'_2) \in \delta((q_1, q_2), \lambda)$ if $\begin{cases} q'_1 \in \delta_1(q_1, a) \text{ and } q'_2 = q_2; \text{ or} \\ q'_2 \in \delta_2(q_2, a) \text{ and } q'_1 = q_1. \end{cases}$

Better notation: $p \xrightarrow{a}_1 q$ for $q \in \delta_1(p, a)$

$$q_1 \xrightarrow{a}_1 q'_1 \quad q_2 \xrightarrow{a}_2 q'_2$$

$$\frac{}{(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)}$$

$$q_1 \xrightarrow{\lambda}_1 q'_1$$

$$\frac{}{(q_1, q_2) \xrightarrow{\lambda} (q'_1, q_2)}$$

$$q_2 \xrightarrow{\lambda}_2 q'_2$$

$$\frac{}{(q_1, q_2) \xrightarrow{\lambda} (q_1, q'_2)}$$

Parallel composition of automata

- Parallel composition of automata is similar to product
- The difference is mostly in the **alphabet of actions**

Parallel composition of automata

- Parallel composition of automata is similar to product
- The difference is mostly in the **alphabet of actions**

With the **product** of automata, the following applied:

- $\Sigma_{M_1 \times M_2} = \Sigma_{M_1} \cap \Sigma_{M_2}$.
- So M_1 must provide all symbols of M_2 .
- So this is not a good **separation of concerns**.

Parallel composition of automata

- Parallel composition of automata is similar to product
- The difference is mostly in the **alphabet of actions**

With the **product** of automata, the following applied:

- $\Sigma_{M_1 \times M_2} = \Sigma_{M_1} \cap \Sigma_{M_2}$.
- So M_1 must provide all symbols of M_2 .
- So this is not a good **separation of concerns**.

With **parallel composition** we separate four types of actions:

- On $\Sigma_1 \cap \Sigma_2$ we still require synchronization, as with product

Parallel composition of automata

- Parallel composition of automata is similar to product
- The difference is mostly in the **alphabet of actions**

With the **product** of automata, the following applied:

- $\Sigma_{M_1 \times M_2} = \Sigma_{M_1} \cap \Sigma_{M_2}$.
- So M_1 must provide all symbols of M_2 .
- So this is not a good **separation of concerns**.

With **parallel composition** we separate four types of actions:

- On $\Sigma_1 \cap \Sigma_2$ we still require synchronization, as with product
- On $\Sigma_1 \setminus \Sigma_2$, M_1 decides what happens (and M_2 does nothing)
- On $\Sigma_2 \setminus \Sigma_1$, M_2 decides what happens (and M_1 does nothing)

Parallel composition of automata

- Parallel composition of automata is similar to product
- The difference is mostly in the **alphabet of actions**

With the **product** of automata, the following applied:

- $\Sigma_{M_1 \times M_2} = \Sigma_{M_1} \cap \Sigma_{M_2}$.
- So M_1 must provide all symbols of M_2 .
- So this is not a good **separation of concerns**.

With **parallel composition** we separate four types of actions:

- On $\Sigma_1 \cap \Sigma_2$ we still require synchronization, as with product
- On $\Sigma_1 \setminus \Sigma_2$, M_1 decides what happens (and M_2 does nothing)
- On $\Sigma_2 \setminus \Sigma_1$, M_2 decides what happens (and M_1 does nothing)
- λ is never synchronized on, instead it is **interleaved**

Parallel Composition of NFA- λ (mathematical)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$

Parallel Composition of NFA- λ (mathematical)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$
- Then $M_1 || M_2 := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2),$

Parallel Composition of NFA- λ (mathematical)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$
- Then $M_1 || M_2 := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$,
- where $\delta((q_1, q_2), a)$ is defined by:

$$\frac{q_1 \xrightarrow{a}_1 q'_1 \quad q_2 \xrightarrow{a}_2 q'_2}{(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)} \quad a \in \Sigma_1 \cap \Sigma_2$$

Parallel Composition of NFA- λ (mathematical)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$
- Then $M_1 || M_2 := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$,
- where $\delta((q_1, q_2), a)$ is defined by:

$$\frac{q_1 \xrightarrow{a}_1 q'_1 \quad q_2 \xrightarrow{a}_2 q'_2}{(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)} \quad a \in \Sigma_1 \cap \Sigma_2$$

$$\frac{q_1 \xrightarrow{a}_1 q'_1}{(q_1, q_2) \xrightarrow{a} (q'_1, q_2)} \quad a \in \Sigma_1 \setminus \Sigma_2 \text{ or } a = \lambda$$

Parallel Composition of NFA- λ (mathematical)

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1^0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, q_2^0, F_2)$
- Then $M_1 || M_2 := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_1^0, q_2^0), F_1 \times F_2)$,
- where $\delta((q_1, q_2), a)$ is defined by:

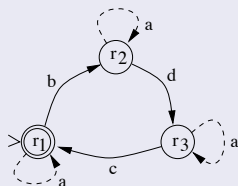
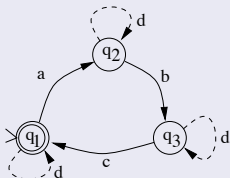
$$\frac{q_1 \xrightarrow{a}_1 q'_1 \quad q_2 \xrightarrow{a}_2 q'_2}{(q_1, q_2) \xrightarrow{a} (q'_1, q'_2)} \quad a \in \Sigma_1 \cap \Sigma_2$$

$$\frac{q_1 \xrightarrow{a}_1 q'_1}{(q_1, q_2) \xrightarrow{a} (q'_1, q_2)} \quad a \in \Sigma_1 \setminus \Sigma_2 \text{ or } a = \lambda$$

$$\frac{q_2 \xrightarrow{a}_2 q'_2}{(q_1, q_2) \xrightarrow{a} (q_1, q'_2)} \quad a \in \Sigma_2 \setminus \Sigma_1 \text{ or } a = \lambda$$

Example parallel composition

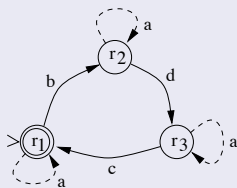
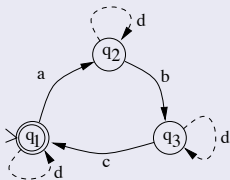
- We now take the parallel composition of the automata:



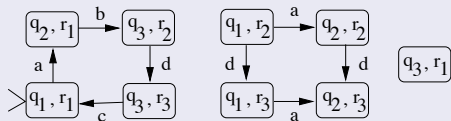
- This results in:

Example parallel composition

- We now take the parallel composition of the automata:

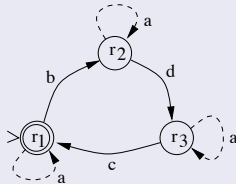
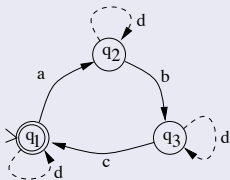


- This results in:

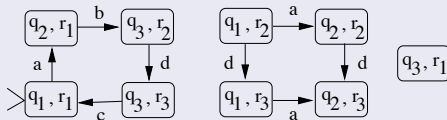


Example parallel composition

- We now take the parallel composition of the automata:



- This results in:



Note:

- only the 1st four states are reachable
- parallel composition and product are identical with “self-loops”

Projection and hiding

Information Hiding

- In software engineering **information hiding** is essential
- In our case we want to **hide** some actions
- Hiding can be done by simply replacing actions by λ

Projection and hiding

Information Hiding

- In software engineering **information hiding** is essential
- In our case we want to **hide** some actions
- Hiding can be done by simply replacing actions by λ

In the previous example:

- we might be only interested in the actions a and d ;
- the actions b and c are meant only as internal synchronization.

Projection and hiding

Information Hiding

- In software engineering **information hiding** is essential
- In our case we want to **hide** some actions
- Hiding can be done by simply replacing actions by λ

In the previous example:

- we might be only interested in the actions a and d ;
- the actions b and c are meant only as internal synchronization.

We can write this in two ways:

$$(\Sigma = \{a, b, c, d\})$$

- $M \upharpoonright \{a, d\}$: project M on the actions a and d
- $M \setminus \{b, c\}$: hide the actions b and c in M (hiding)

Projection and hiding

Information Hiding

- In software engineering **information hiding** is essential
- In our case we want to **hide** some actions
- Hiding can be done by simply replacing actions by λ

In the previous example:

- we might be only interested in the actions a and d ;
- the actions b and c are meant only as internal synchronization.

We can write this in two ways:

$$(\Sigma = \{a, b, c, d\})$$

- $M \upharpoonright \{a, d\}$: project M on the actions a and d
- $M \setminus \{b, c\}$: hide the actions b and c in M (hiding)
- The effect of both operations is the same:
 b and c are replaced by a λ

Projection and hiding

Information Hiding

- In software engineering **information hiding** is essential
- In our case we want to **hide** some actions
- Hiding can be done by simply replacing actions by λ

In the previous example:

- we might be only interested in the actions a and d ;
- the actions b and c are meant only as internal synchronization.

We can write this in two ways:

$$(\Sigma = \{a, b, c, d\})$$

- $M \upharpoonright \{a, d\}$: project M on the actions a and d
- $M \setminus \{b, c\}$: hide the actions b and c in M (hiding)
- The effect of both operations is the same:
 b and c are replaced by a λ (that can be removed)

Projection for Languages

Projection for **symbols** $a \in \Sigma$:

- $a \upharpoonright \Sigma_1 = \begin{cases} a, & \text{als } a \in \Sigma_1 \\ \lambda, & \text{als } a \notin \Sigma_1 \end{cases}$

Projection for Languages

Projection for **symbols** $a \in \Sigma$:

- $a \upharpoonright \Sigma_1 = \begin{cases} a, & \text{als } a \in \Sigma_1 \\ \lambda, & \text{als } a \notin \Sigma_1 \end{cases}$

Projection for **words** $w \in \Sigma^*$:

- $\lambda \upharpoonright \Sigma_1 = \lambda$
- $(aw) \upharpoonright \Sigma_1 = (a \upharpoonright \Sigma_1)(w \upharpoonright \Sigma_1)$

Projection for Languages

Projection for **symbols** $a \in \Sigma$:

- $a \upharpoonright \Sigma_1 = \begin{cases} a, & \text{als } a \in \Sigma_1 \\ \lambda, & \text{als } a \notin \Sigma_1 \end{cases}$

Projection for **words** $w \in \Sigma^*$:

- $\lambda \upharpoonright \Sigma_1 = \lambda$
- $(aw) \upharpoonright \Sigma_1 = (a \upharpoonright \Sigma_1)(w \upharpoonright \Sigma_1)$
- Example: $abbccdaacadbba \upharpoonright \{a, c\} = accaaca$

Projection for Languages

Projection for **symbols** $a \in \Sigma$:

- $a \upharpoonright \Sigma_1 = \begin{cases} a, & \text{als } a \in \Sigma_1 \\ \lambda, & \text{als } a \notin \Sigma_1 \end{cases}$

Projection for **words** $w \in \Sigma^*$:

- $\lambda \upharpoonright \Sigma_1 = \lambda$
- $(aw) \upharpoonright \Sigma_1 = (a \upharpoonright \Sigma_1)(w \upharpoonright \Sigma_1)$
- Example: $abbccdaacadbba \upharpoonright \{a, c\} = accaaca$

Projection for **languages** $L \subseteq \Sigma^*$:

- $L \upharpoonright \Sigma_1 = \{w \upharpoonright \Sigma_1 \mid w \in L\}$

Projection for Languages

Projection for **symbols** $a \in \Sigma$:

- $a \upharpoonright \Sigma_1 = \begin{cases} a, & \text{als } a \in \Sigma_1 \\ \lambda, & \text{als } a \notin \Sigma_1 \end{cases}$

Projection for **words** $w \in \Sigma^*$:

- $\lambda \upharpoonright \Sigma_1 = \lambda$
- $(aw) \upharpoonright \Sigma_1 = (a \upharpoonright \Sigma_1)(w \upharpoonright \Sigma_1)$
- Example: $abbccdaacadbba \upharpoonright \{a, c\} = accaaca$

Projection for **languages** $L \subseteq \Sigma^*$:

- $L \upharpoonright \Sigma_1 = \{w \upharpoonright \Sigma_1 \mid w \in L\}$

Projection for **automata** M (NFA- λ):

- $M \upharpoonright \Sigma_1$: Replace every occurrence of $a \notin \Sigma_1$ by λ

Language of parallel composition and projection

- Let NFA- λ M_1 and M_2 be given, with alphabet Σ_1 and Σ_2

Language of parallel composition and projection

- Let NFA- λ M_1 and M_2 be given, with alphabet Σ_1 and Σ_2
- We already saw for the product automata:

$$\mathcal{L}(M_1 \times M_2) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$$

- So:
 - $w \in \mathcal{L}(M_1 \times M_2)$ if and only if
 - $w \in \mathcal{L}(M_1)$ and $w \in \mathcal{L}(M_2)$

Language of parallel composition and projection

- Let NFA- λ M_1 and M_2 be given, with alphabet Σ_1 and Σ_2
- We already saw for the product automata:
$$\mathcal{L}(M_1 \times M_2) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$$
- So:
 - $w \in \mathcal{L}(M_1 \times M_2)$ if and only if
 - $w \in \mathcal{L}(M_1)$ and $w \in \mathcal{L}(M_2)$
- What is the language of parallel composition? $\mathcal{L}(M_1 || M_2)$

Language of parallel composition and projection

- Let NFA- λ M_1 and M_2 be given, with alphabet Σ_1 and Σ_2
- We already saw for the product automata:

$$\mathcal{L}(M_1 \times M_2) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$$
- So:
 - $w \in \mathcal{L}(M_1 \times M_2)$ if and only if
 - $w \in \mathcal{L}(M_1)$ and $w \in \mathcal{L}(M_2)$
- What is the language of parallel composition? $\mathcal{L}(M_1 || M_2)$
- The following applies:
 - $w \in \mathcal{L}(M_1 || M_2)$ if and only if
 - $w \upharpoonright \Sigma_1 \in \mathcal{L}(M_1)$ and $w \upharpoonright \Sigma_2 \in \mathcal{L}(M_2)$

System Verification by Model Checking

Modeling and Specification

- Let Ext be the alphabet of external symbols; Int is the alphabet of internal communications
- Let $Spec$ be a RE over alphabet Ext
- Let $Comp_i$ be components over alphabet $Ext \cup Int$.

Correctness: The external behaviour of the composed system conforms to the specification,

$$\mathcal{L}((Comp_1 \parallel \dots \parallel Comp_n) \setminus Int) \subseteq \mathcal{L}(Spec)$$

Practical challenge: the intermediate state space grows exponentially with the number of components.

After hiding and minimisation, the result is typically small, but it is not (always) possible to compute the small automaton directly.

Contents

- 1 Composition of Automata in Software Engineering
 - Product of Automata
 - Parallel composition
 - Projection and Hiding
- 2 Grammars and Derivations
 - Derivations
 - Derivation trees
 - Generated Language
- 3 Regular grammars
- 4 Syntax of programming languages
 - Ambiguity
 - Naive Parsing

Example: calculator expressions

Alphabet: $\{1, 2, 3, +, \times\}$. **Variables:** $\{S, T\}$. **Expression:** $1 + 2 \times 3$.

Let us specify the **language of all correct expressions**:

Example: calculator expressions

Alphabet: $\{1, 2, 3, +, \times\}$. **Variables:** $\{S, T\}$. **Expression:** $1 + 2 \times 3$.

Let us specify the **language of all correct expressions**:

Grammar G_1

$$S \rightarrow 1 \mid 2 \mid 3 \mid S + S \mid S \times S$$

Example: calculator expressions

Alphabet: $\{1, 2, 3, +, \times\}$. **Variables:** $\{S, T\}$. **Expression:** $1 + 2 \times 3$.

Let us specify the **language of all correct expressions**:

Grammar G_1

$$S \rightarrow 1 \mid 2 \mid 3 \mid S + S \mid S \times S$$

Grammar G_2

$$S \rightarrow T \times S \mid T$$

$$T \rightarrow 1 \mid 2 \mid 3 \mid T + T$$

Example: calculator expressions

Alphabet: $\{1, 2, 3, +, \times\}$. **Variables:** $\{S, T\}$. **Expression:** $1 + 2 \times 3$.

Let us specify the **language of all correct expressions**:

Grammar G_1

$$S \rightarrow 1 \mid 2 \mid 3 \mid S + S \mid S \times S$$

Grammar G_2

$$S \rightarrow T \times S \mid T$$
$$T \rightarrow 1 \mid 2 \mid 3 \mid T + T$$

Grammar G_3

$$S \rightarrow T + S \mid T$$
$$T \rightarrow 1 \mid 2 \mid 3 \mid T \times T$$

Example: calculator expressions

Alphabet: $\{1, 2, 3, +, \times\}$. **Variables:** $\{S, T\}$. **Expression:** $1 + 2 \times 3$.

Let us specify the **language of all correct expressions**:

Grammar G_1

$$S \rightarrow 1 \mid 2 \mid 3 \mid S + S \mid S \times S$$

Grammar G_2

$$S \rightarrow T \times S \mid T$$

$$T \rightarrow 1 \mid 2 \mid 3 \mid T + T$$

Grammar G_3

$$S \rightarrow T + S \mid T$$

$$T \rightarrow 1 \mid 2 \mid 3 \mid T \times T$$

Only G_3 reflects the “correct” arithmetic interpretation

Note: All grammars are still ambiguous (check!)

Basic Concepts of Context-Free Grammars

Definition

Context-Free Grammar (CFG): 4-tuple $\langle V, \Sigma, P, S \rangle$ with

- V a set of *variables* (help symbols, *non-terminals*)
- Σ (again) the *alphabet*, disjunct with V (so $V \cap \Sigma = \emptyset$)
- $P \subseteq V \times (V \cup \Sigma)^*$ a set of *rules* (production)
- $S \in V$ the *start symbol*

Basic Concepts of Context-Free Grammars

Definition

Context-Free Grammar (CFG): 4-tuple $\langle V, \Sigma, P, S \rangle$ with

- V a set of *variables* (help symbols, *non-terminals*)
- Σ (again) the *alphabet*, disjoint with V (so $V \cap \Sigma = \emptyset$)
- $P \subseteq V \times (V \cup \Sigma)^*$ a set of *rules* (production)
- $S \in V$ the *start symbol*

Additional remarks

- A rule is a pair (A, w) with $A \in V$ and $w \in (V \cup \Sigma)^*$
 - Notation: $A \rightarrow w_1 \mid w_2 \mid \dots$ if $(A, w_1), (A, w_2), \dots \in P$.
 - Special case: *null-rule* $A \rightarrow \lambda$
- Convention:
 - Variables with upper case letters (A, B, \dots)
 - Alphabet with lower case letters (a, b, \dots)

Examples of CFG's

- $V = \{S, A\}$ and $\Sigma = \{a, b\}$:

$$S \rightarrow AA$$
$$A \rightarrow AAA \mid bA \mid Ab \mid a$$

Examples of CFG's

- $V = \{S, A\}$ and $\Sigma = \{a, b\}$:

$$S \rightarrow AA$$

$$A \rightarrow AAA \mid bA \mid Ab \mid a$$

- $V = \{\textit{Sentence}, \textit{NounPhrase}, \dots\}$ and $\Sigma = \{a, \dots, z\}^*$:

$$\textit{Sentence} \rightarrow \textit{NounPhrase} \textit{VerbPhrase} \mid$$

$$\textit{NounPhrase} \textit{Verb} \textit{D.O.Phrase}$$

$$\textit{NounPhrase} \rightarrow \textit{ProperNoun} \mid$$

$$\textit{Determiner} \textit{CommonNoun}$$

$$\textit{Determiner} \rightarrow a \mid the$$

$$\textit{CommonNoun} \rightarrow car \mid hamburger \mid \dots$$

...

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow uwv$$

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow u w v$$

- Examples: rules (i) $A \rightarrow Aa$ and (ii) $A \rightarrow b$:

$$bA$$

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow u w v$$

- Examples: rules (i) $A \rightarrow Aa$ and (ii) $A \rightarrow b$:

$$bA \xrightarrow{(i)} bAa$$

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow uwv$$

- Examples: rules (i) $A \rightarrow Aa$ and (ii) $A \rightarrow b$:

$$bA \xrightarrow{(i)} bAa \xrightarrow{(i)} bAaa$$

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow uwv$$

- Examples: rules (i) $A \rightarrow Aa$ and (ii) $A \rightarrow b$:

$$bA \xrightarrow{(i)} bAa \xrightarrow{(i)} bAaa \xrightarrow{(ii)} bbbaa$$

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow uwv$$

- Examples: rules (i) $A \rightarrow Aa$ and (ii) $A \rightarrow b$:

$$bA \xrightarrow{(i)} bAa \xrightarrow{(i)} bAaa \xrightarrow{(ii)} bbaa$$
$$bA \xrightarrow{(i)}$$

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow u w v$$

- Examples: rules (i) $A \rightarrow Aa$ and (ii) $A \rightarrow b$:

$$bA \xrightarrow{(i)} bAa \xrightarrow{(i)} bAaa \xrightarrow{(ii)} bbaa$$

$$bA \xrightarrow{(i)} bAa \xrightarrow{(ii)}$$

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow uwv$$

- Examples: rules (i) $A \rightarrow Aa$ and (ii) $A \rightarrow b$:

$$bA \xrightarrow{(i)} bAa \xrightarrow{(i)} bAaa \xrightarrow{(ii)} bbbaa$$

$$bA \xrightarrow{(i)} bAa \xrightarrow{(ii)} bba$$

Rule application

- A rule of the form

$$A \rightarrow w$$

can be *applied* to words of the form

$$uAv$$

and results in a *derivation* of the form

$$uAv \Rightarrow uwv$$

- Examples: rules (i) $A \rightarrow Aa$ and (ii) $A \rightarrow b$:

$$bA \xrightarrow{(i)} bAa \xrightarrow{(i)} bAaa \xrightarrow{(ii)} bb aa$$

$$bA \xrightarrow{(i)} bAa \xrightarrow{(ii)} bba$$

$$bA \xrightarrow{(ii)} bb$$

Derivability

Definition

Take a CFG $\langle V, \Sigma, P, S \rangle$ and a string $v \in (V \cup \Sigma)^*$.

The set of *derivable strings* $Der(v)$ is the *smallest* set such that

- $v \in Der(v)$;
- If $xAy \in Der(v)$, and $(A \rightarrow w) \in P$, then $xwy \in Der(v)$.

Derivability

Definition

Take a CFG $\langle V, \Sigma, P, S \rangle$ and a string $v \in (V \cup \Sigma)^*$.

The set of *derivable strings* $Der(v)$ is the *smallest* set such that

- $v \in Der(v)$;
- If $xAy \in Der(v)$, and $(A \rightarrow w) \in P$, then $xwy \in Der(v)$.

So: $w \in Der(v)$ if a (possibly empty) series of derivations exists:

$$v = w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \dots \Rightarrow w_n = w$$

Notation: $v \xRightarrow{*} w$ for “ w is derivable from v ”

Derivability

Definition

Take a CFG $\langle V, \Sigma, P, S \rangle$ and a string $v \in (V \cup \Sigma)^*$.

The set of *derivable strings* $Der(v)$ is the *smallest* set such that

- $v \in Der(v)$;
- If $x A y \in Der(v)$, and $(A \rightarrow w) \in P$, then $x w y \in Der(v)$.

So: $w \in Der(v)$ if a (possibly empty) series of derivations exists:

$$v = w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \dots \Rightarrow w_n = w$$

Notation: $v \xRightarrow{*} w$ for “ w is derivable from v ”

Example: $b A \xRightarrow{*} b b a a$ and $b A \xRightarrow{*} b b$.

So $b b \in Der(b A)$.

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)
 $S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow ab aAA$
 $\Rightarrow ab abAA$
 $\Rightarrow ab ab aA$
 $\Rightarrow ab ab a a$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)

$S \Rightarrow AA$

$\Rightarrow aA$

$\Rightarrow aAAA$

$\Rightarrow abAAA$

$\Rightarrow ab aAA$

$\Rightarrow ab abAA$

$\Rightarrow ab abaA$

$\Rightarrow ab abaa$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)
 $S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow ab aAA$
 $\Rightarrow ab abAA$
 $\Rightarrow ab ab aA$
 $\Rightarrow ab ab a a$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)
 $S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow ab aAA$
 $\Rightarrow ab abAA$
 $\Rightarrow ab abaA$
 $\Rightarrow ab abaa$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)

$$\begin{aligned}
 S &\Rightarrow AA \\
 &\Rightarrow aA \\
 &\Rightarrow aAA \\
 &\Rightarrow abAAA \\
 &\Rightarrow ababAA \\
 &\Rightarrow abababAA \\
 &\Rightarrow ababababA \\
 &\Rightarrow ababababaa
 \end{aligned}$$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)
 $S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAA$
 $\Rightarrow abAAA$
 $\Rightarrow ab aAA$
 $\Rightarrow ab abAA$
 $\Rightarrow ab abaA$
 $\Rightarrow ab abaa$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)

$S \Rightarrow AA$

$\Rightarrow aA$

$\Rightarrow aAA$

$\Rightarrow abAAA$

$\Rightarrow ababAAA$

$\Rightarrow abababAAA$

$\Rightarrow ababababA$

$\Rightarrow ababababaa$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)

$$\begin{aligned}
 S &\Rightarrow AA \\
 &\Rightarrow aA \\
 &\Rightarrow aAA \\
 &\Rightarrow abAA \\
 &\Rightarrow ab aAA \\
 &\Rightarrow ab abAA \\
 &\Rightarrow ab ab aA \\
 &\Rightarrow ab ab a a
 \end{aligned}$$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)

$S \Rightarrow AA$

$\Rightarrow aA$

$\Rightarrow aAAA$

$\Rightarrow abAAA$

$\Rightarrow ababAA$

$\Rightarrow ababaA$

$\Rightarrow ababaa$

(ii)

$S \Rightarrow AA$

$\Rightarrow AAAA$

$\Rightarrow aAAA$

$\Rightarrow abAAA$

$\Rightarrow ababAA$

$\Rightarrow ababaA$

$\Rightarrow ababaa$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAA$
 $\Rightarrow abAAA$
 $\Rightarrow abaAA$
 $\Rightarrow ababAA$
 $\Rightarrow ababaA$
 $\Rightarrow ababaa$

(ii)

$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow abaAA$
 $\Rightarrow ababAA$
 $\Rightarrow ababaA$
 $\Rightarrow ababaa$

(iii)

$S \Rightarrow AA$
 $\Rightarrow Aa$
 $\Rightarrow AAAa$
 $\Rightarrow AAbAa$
 $\Rightarrow AAbaa$
 $\Rightarrow AbAbaa$
 $\Rightarrow Ababaa$
 $\Rightarrow ababaa$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow abaAA$
 $\Rightarrow ababAA$
 $\Rightarrow ababaA$
 $\Rightarrow ababaa$

(ii)

$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow abaAA$
 $\Rightarrow ababAA$
 $\Rightarrow ababaA$
 $\Rightarrow ababaa$

(iii)

$S \Rightarrow AA$
 $\Rightarrow Aa$
 $\Rightarrow AAAa$
 $\Rightarrow AAbAa$
 $\Rightarrow AAbaa$
 $\Rightarrow AbAbaa$
 $\Rightarrow Ababaa$
 $\Rightarrow ababaa$

(iv)

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow aAAa$
 $\Rightarrow abAAa$
 $\Rightarrow abAbAa$
 $\Rightarrow ababAa$
 $\Rightarrow ababaa$

Examples of derivations

Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$

(i)

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow abaAA$
 $\Rightarrow ababAA$
 $\Rightarrow ababaA$
 $\Rightarrow ababaa$

(ii)

$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow abaAA$
 $\Rightarrow ababAA$
 $\Rightarrow ababaA$
 $\Rightarrow ababaa$

(iii)

$S \Rightarrow AA$
 $\Rightarrow Aa$
 $\Rightarrow AAAa$
 $\Rightarrow AAbAa$
 $\Rightarrow AAbaa$
 $\Rightarrow AbAbaa$
 $\Rightarrow Ababaa$
 $\Rightarrow ababaa$

(iv)

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow aAAa$
 $\Rightarrow abAAa$
 $\Rightarrow abAbAa$
 $\Rightarrow ababAa$
 $\Rightarrow ababaa$

- Leftmost derivations: (i) and (ii) (left variable first)
- Rightmost derivations: (iii) (right variable first)

Derivation trees

Definition derivation tree

Take a CFG $G = \langle V, \Sigma, P, S \rangle$ and a derivation $S \xRightarrow{*} w$.

The (ordered) derivation tree of $S \xRightarrow{*} w$ is constructed as follows:

- Select S as root
- At the application of a rule $A \rightarrow x_1 x_2 \cdots x_n$ (all $x_i \in V \cup \Sigma$):
add x_1, x_2, \dots, x_n as children of A
- At the application of a rule $A \rightarrow \lambda$:
add λ as (only) child of A

Derivation trees

Definition derivation tree

Take a CFG $G = \langle V, \Sigma, P, S \rangle$ and a derivation $S \xRightarrow{*} w$.

The (ordered) derivation tree of $S \xRightarrow{*} w$ is constructed as follows:

- Select S as root
- At the application of a rule $A \rightarrow x_1 x_2 \cdots x_n$ (all $x_i \in V \cup \Sigma$):
add x_1, x_2, \dots, x_n as children of A
- At the application of a rule $A \rightarrow \lambda$:
add λ as (only) child of A

Remark

At any time during the construction, the derived word is equal to the symbols on the leaves, in the order of the tree

Examples of derivation trees (1)

Leftmost derivation

$$\begin{aligned}
 S &\Rightarrow AA \\
 &\Rightarrow aA \\
 &\Rightarrow aAAA \\
 &\Rightarrow abAAA \\
 &\Rightarrow ab aAA \\
 &\Rightarrow ab abAA \\
 &\Rightarrow ab ab aA \\
 &\Rightarrow ab ab a a
 \end{aligned}$$

Derivation tree

$$S$$

Rightmost derivation

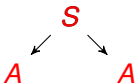
$$\begin{aligned}
 S &\Rightarrow A A \\
 &\Rightarrow A A A A \\
 &\Rightarrow A A A a \\
 &\Rightarrow A A b A a \\
 &\Rightarrow A A b a a \\
 &\Rightarrow A b A b a a \\
 &\Rightarrow A b a b a a \\
 &\Rightarrow a b a b a a
 \end{aligned}$$

Examples of derivation trees (1)

Leftmost derivation

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow ab aAA$
 $\Rightarrow ab abAA$
 $\Rightarrow ab ab aA$
 $\Rightarrow ab ab a a$

Derivation tree



Rightmost derivation

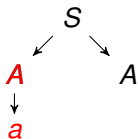
$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow AA A a$
 $\Rightarrow AA b A a$
 $\Rightarrow A A b a a$
 $\Rightarrow A b A b a a$
 $\Rightarrow A b a b a a$
 $\Rightarrow a b a b a a$

Examples of derivation trees (1)

Leftmost derivation

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow ab aAA$
 $\Rightarrow ab abAA$
 $\Rightarrow ab ab aA$
 $\Rightarrow ab ab a a$

Derivation tree



Rightmost derivation

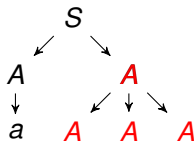
$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow AA A a$
 $\Rightarrow A A b A a$
 $\Rightarrow A A b a a$
 $\Rightarrow A b A b a a$
 $\Rightarrow A b a b a a$
 $\Rightarrow a b a b a a$

Examples of derivation trees (1)

Leftmost derivation

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow ab aAA$
 $\Rightarrow ab abAA$
 $\Rightarrow ab abaA$
 $\Rightarrow ab abaa$

Derivation tree



Rightmost derivation

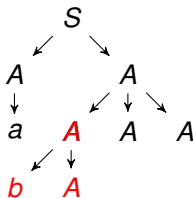
$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow AA Aa$
 $\Rightarrow AAb Aa$
 $\Rightarrow A A b a a$
 $\Rightarrow Ab A b a a$
 $\Rightarrow A b a b a a$
 $\Rightarrow ab abaa$

Examples of derivation trees (1)

Leftmost derivation

$$\begin{aligned}
 S &\Rightarrow AA \\
 &\Rightarrow aA \\
 &\Rightarrow aAA \\
 &\Rightarrow abAA \\
 &\Rightarrow ab aAA \\
 &\Rightarrow ab abAA \\
 &\Rightarrow ab ab aA \\
 &\Rightarrow ab ab a a
 \end{aligned}$$

Derivation tree



Rightmost derivation

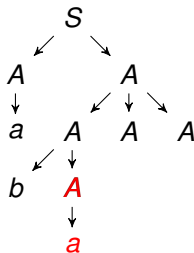
$$\begin{aligned}
 S &\Rightarrow AA \\
 &\Rightarrow AAAA \\
 &\Rightarrow AA A a \\
 &\Rightarrow AA b A a \\
 &\Rightarrow A A b a a \\
 &\Rightarrow A b A b a a \\
 &\Rightarrow A b a b a a \\
 &\Rightarrow a b a b a a
 \end{aligned}$$

Examples of derivation trees (1)

Leftmost derivation

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAA$
 $\Rightarrow abAA$
 $\Rightarrow ab aAA$
 $\Rightarrow ab abAA$
 $\Rightarrow ab abaA$
 $\Rightarrow ab abaa$

Derivation tree



Rightmost derivation

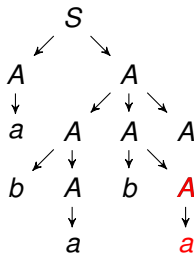
$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow AA Aa$
 $\Rightarrow AAb Aa$
 $\Rightarrow A A b a a$
 $\Rightarrow Ab A b a a$
 $\Rightarrow A b a b a a$
 $\Rightarrow ab abaa$

Examples of derivation trees (1)

Leftmost derivation

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAA$
 $\Rightarrow abAA$
 $\Rightarrow abaAA$
 $\Rightarrow ababAA$
 $\Rightarrow abab aA$
 $\Rightarrow ababaa$

Derivation tree



Rightmost derivation

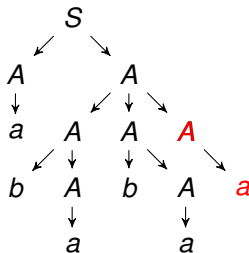
$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow AAaA$
 $\Rightarrow AAbAa$
 $\Rightarrow AAbaa$
 $\Rightarrow AbAbaa$
 $\Rightarrow Ababaa$
 $\Rightarrow ababaa$

Examples of derivation trees (1)

Leftmost derivation

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAA$
 $\Rightarrow abAA$
 $\Rightarrow abAa$
 $\Rightarrow ababAA$
 $\Rightarrow ababaA$
 $\Rightarrow ababaa$

Derivation tree



Rightmost derivation

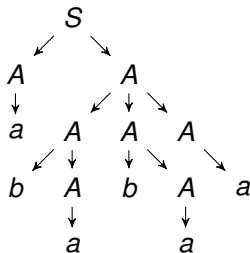
$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow AAaA$
 $\Rightarrow AAbAa$
 $\Rightarrow AAbaa$
 $\Rightarrow AbAbaa$
 $\Rightarrow Ababaa$
 $\Rightarrow ababaa$

Examples of derivation trees (1)

Leftmost derivation

$S \Rightarrow AA$
 $\Rightarrow aA$
 $\Rightarrow aAAA$
 $\Rightarrow abAAA$
 $\Rightarrow abAaA$
 $\Rightarrow ababAaA$
 $\Rightarrow ababAa$
 $\Rightarrow ababaa$

Derivation tree



Rightmost derivation

$S \Rightarrow AA$
 $\Rightarrow AAAA$
 $\Rightarrow AAaA$
 $\Rightarrow AAbAa$
 $\Rightarrow AAbaa$
 $\Rightarrow AbAbaa$
 $\Rightarrow Ababaa$
 $\Rightarrow ababaa$

- The derivation tree makes no distinction based on the *order*
- One-to-one relation between leftmost derivations and derivation trees

Examples of derivation trees (2)

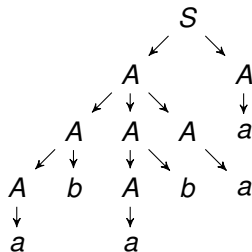
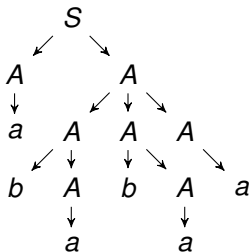
- Derivations of a single sentence can result in different trees

Examples of derivation trees (2)

- Derivations of a single sentence can result in different trees
- Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$
- Multiple derivation trees exist for $S \xRightarrow{*} ababaa$
(so also different leftmost derivations)

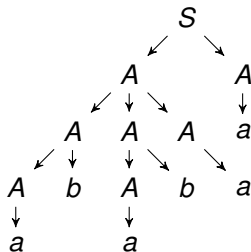
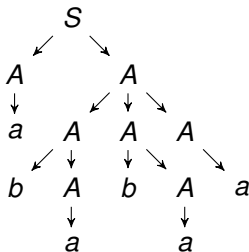
Examples of derivation trees (2)

- Derivations of a single sentence can result in different trees
- Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$
- Multiple derivation trees exist for $S \xRightarrow{*} ababaa$
(so also different leftmost derivations)



Examples of derivation trees (2)

- Derivations of a single sentence can result in different trees
- Rules: $S \rightarrow AA$ and $A \rightarrow AAA \mid bA \mid Ab \mid a$
- Multiple derivation trees exist for $S \xRightarrow{*} ababaa$
(so also different leftmost derivations)



- Such a grammar is called *ambiguous*

Sentential forms, sentences and language

Definition: language of a context-free grammar

Given a CFG $G = \langle V, \Sigma, P, S \rangle$:

- If $S \xRightarrow{*} w$, then w is called a *sentential form* (in general: $w \in (V \cup \Sigma)^*$)
- If $w \in \Sigma^*$, then w is called a *sentence*.
- The *language* of G is defined as $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \xRightarrow{*} w\}$.

Sentential forms, sentences and language

Definition: language of a context-free grammar

Given a CFG $G = \langle V, \Sigma, P, S \rangle$:

- If $S \xRightarrow{*} w$, then w is called a *sentential form* (in general: $w \in (V \cup \Sigma)^*$)
- If $w \in \Sigma^*$, then w is called a *sentence*.
- The *language* of G is defined as $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \xRightarrow{*} w\}$.

Definition: Context-free language

A set of words $X \subseteq \Sigma^*$ is a *context-free language* if a CFG G exists, such that $X = \mathcal{L}(G)$.

We sometimes write **CFG** for the *class of context-free languages*.

Examples of context-free languages

- 1 Number of a 's followed by the same number of b 's?
($\{a^n b^n \mid n \geq 0\}$)
- 2 Number of a 's, then b 's, then the same number of a 's?
($\{a^n b^m a^n \mid n, m > 0\}$)
- 3 Correctly nested brackets? (a is left bracket, b right bracket)
- 4 Palindromes over a, b ?

Examples of context-free languages

- 1 Number of a 's followed by the same number of b 's?
($\{a^n b^n \mid n \geq 0\}$)

$$S \rightarrow a S b \mid \lambda$$

- 2 Number of a 's, then b 's, then the same number of a 's?
($\{a^n b^m a^n \mid n, m > 0\}$)

- 3 Correctly nested brackets? (a is left bracket, b right bracket)

- 4 Palindromes over a, b ?

Examples of context-free languages

- 1 Number of a 's followed by the same number of b 's?
 ($\{a^n b^n \mid n \geq 0\}$)

$$S \rightarrow a S b \mid \lambda$$

- 2 Number of a 's, then b 's, then the same number of a 's?
 ($\{a^n b^m a^n \mid n, m > 0\}$)

$$S \rightarrow a S a \mid a A a \quad A \rightarrow b A \mid b$$

- 3 Correctly nested brackets? (a is left bracket, b right bracket)

- 4 Palindromes over a, b ?

Examples of context-free languages

- 1 Number of a 's followed by the same number of b 's?
 ($\{a^n b^n \mid n \geq 0\}$)

$$S \rightarrow a S b \mid \lambda$$

- 2 Number of a 's, then b 's, then the same number of a 's?
 ($\{a^n b^m a^n \mid n, m > 0\}$)

$$S \rightarrow a S a \mid a A a \quad A \rightarrow b A \mid b$$

- 3 Correctly nested brackets? (a is left bracket, b right bracket)

$$S \rightarrow S S \mid a S b \mid \lambda$$

- 4 Palindromes over a, b ?

Examples of context-free languages

- 1 Number of a 's followed by the same number of b 's?
 ($\{a^n b^n \mid n \geq 0\}$)

$$S \rightarrow a S b \mid \lambda$$

- 2 Number of a 's, then b 's, then the same number of a 's?
 ($\{a^n b^m a^n \mid n, m > 0\}$)

$$S \rightarrow a S a \mid a A a \quad A \rightarrow b A \mid b$$

- 3 Correctly nested brackets? (a is left bracket, b right bracket)

$$S \rightarrow S S \mid a S b \mid \lambda$$

- 4 Palindromes over a, b ?

$$S \rightarrow a S a \mid b S b \mid a \mid b \mid \lambda$$

These languages are all *non-regular*!

Considerations about context-free languages

- Which languages are generated by the following grammars?

$$G_1 : \quad S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid \lambda$$

$$G_2 : \quad S \rightarrow aS \mid aB \quad B \rightarrow bB \mid \lambda$$

Considerations about context-free languages

- Which languages are generated by the following grammars?

$$G_1 : \quad S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid \lambda$$

$$G_2 : \quad S \rightarrow aS \mid aB \quad B \rightarrow bB \mid \lambda$$

Answer: both generate a^+b^* (a regular language!).

Considerations about context-free languages

- Which languages are generated by the following grammars?

$$G_1 : \quad S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid \lambda$$

$$G_2 : \quad S \rightarrow aS \mid aB \quad B \rightarrow bB \mid \lambda$$

Answer: both generate a^+b^* (a regular language!).

- Are all regular languages context-free? How can you prove something like that?

Considerations about context-free languages

- Which languages are generated by the following grammars?

$$G_1 : \quad S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid \lambda$$

$$G_2 : \quad S \rightarrow aS \mid aB \quad B \rightarrow bB \mid \lambda$$

Answer: both generate a^+b^* (a regular language!).

- Are all regular languages context-free? How can you prove something like that?
- Can you give an **unambiguous** grammar for every context-free language?

Considerations about context-free languages

- Which languages are generated by the following grammars?

$$G_1 : \quad S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid \lambda$$

$$G_2 : \quad S \rightarrow aS \mid aB \quad B \rightarrow bB \mid \lambda$$

Answer: both generate a^+b^* (a regular language!).

- Are all regular languages context-free? How can you prove something like that?
- Can you give an **unambiguous** grammar for every context-free language?
- G_1 and G_2 are called *equivalent* if $L(G_1) = L(G_2)$.
Can we always check equivalence of CFGs? How?

Contents

- 1 Composition of Automata in Software Engineering
 - Product of Automata
 - Parallel composition
 - Projection and Hiding
- 2 Grammars and Derivations
 - Derivations
 - Derivation trees
 - Generated Language
- 3 Regular grammars
- 4 Syntax of programming languages
 - Ambiguity
 - Naive Parsing

Regular grammars

Definition: Regular grammar

A grammar is called *regular* if every rule has one of the following forms (with $A, B \in V$ and $a \in \Sigma$):

- 1 $A \rightarrow a$
- 2 $A \rightarrow aB$
- 3 $A \rightarrow \lambda$

Regular grammars

Definition: Regular grammar

A grammar is called *regular* if every rule has one of the following forms (with $A, B \in V$ and $a \in \Sigma$):

- 1 $A \rightarrow a$
- 2 $A \rightarrow aB$
- 3 $A \rightarrow \lambda$

Remarks

- If G_1 and G_2 are equivalent, and G_2 is regular, then G_1 does not have to be regular
- Example: $G_1 : S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid \lambda$
 $G_2 : S \rightarrow aS \mid aB \quad B \rightarrow bB \mid \lambda$

Regular grammars

Definition: Regular grammar

A grammar is called *regular* if every rule has one of the following forms (with $A, B \in V$ and $a \in \Sigma$):

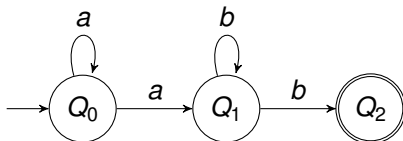
- 1 $A \rightarrow a$
- 2 $A \rightarrow aB$
- 3 $A \rightarrow \lambda$

Remarks

- If G_1 and G_2 are equivalent, and G_2 is regular, then G_1 does not have to be regular
- Example: $G_1 : S \rightarrow AB \quad A \rightarrow aA \mid a \quad B \rightarrow bB \mid \lambda$
 $G_2 : S \rightarrow aS \mid aB \quad B \rightarrow bB \mid \lambda$
 (see previous sheet; both have language a^+b^*)

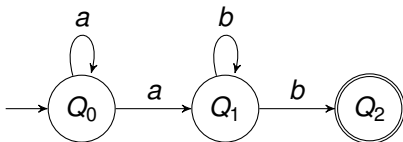
Finite automaton to grammar: Example

- Automaton M :



Finite automaton to grammar: Example

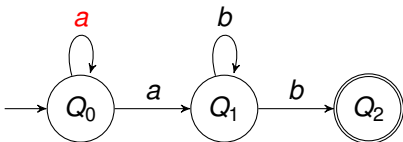
- Automaton M :



- Grammar G :
 $Q_0 \rightarrow$
 $Q_1 \rightarrow$
 $Q_2 \rightarrow$

Finite automaton to grammar: Example

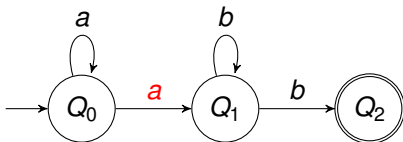
- Automaton M :



- Grammar G :
 $Q_0 \rightarrow a Q_0$
 $Q_1 \rightarrow$
 $Q_2 \rightarrow$

Finite automaton to grammar: Example

- Automaton M :

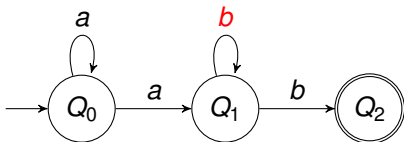


- Grammar G :

Q_0	\rightarrow	$a Q_0 \mid a Q_1$
Q_1	\rightarrow	
Q_2	\rightarrow	

Finite automaton to grammar: Example

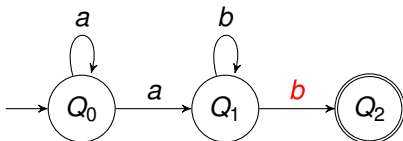
- Automaton M :



- Grammar G :
 $Q_0 \rightarrow a Q_0 \mid a Q_1$
 $Q_1 \rightarrow b Q_1$
 $Q_2 \rightarrow$

Finite automaton to grammar: Example

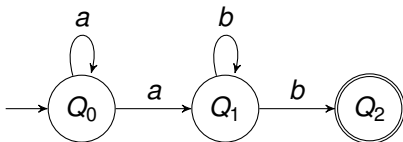
- Automaton M :



- Grammar G :
 $Q_0 \rightarrow a Q_0 \mid a Q_1$
 $Q_1 \rightarrow b Q_1 \mid b Q_2$
 $Q_2 \rightarrow$

Finite automaton to grammar: Example

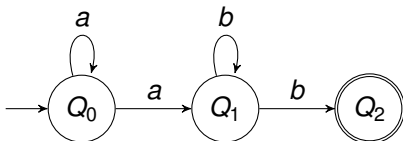
- Automaton M :



- Grammar G :
 $Q_0 \rightarrow aQ_0 \mid aQ_1$
 $Q_1 \rightarrow bQ_1 \mid bQ_2$
 $Q_2 \rightarrow \lambda$

Finite automaton to grammar: Example

- Automaton M :



- Grammar G :

$$Q_0 \rightarrow aQ_0 \mid aQ_1$$

$$Q_1 \rightarrow bQ_1 \mid bQ_2$$

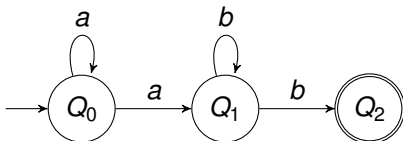
$$Q_2 \rightarrow \lambda$$

- Common word $aabb \in L(M) \cap L(G)$:

Computation	Derivation	Partial word
$[Q_0, aabb] \vdash [Q_0, abb]$	$Q_0 \Rightarrow aQ_0$	a

Finite automaton to grammar: Example

- Automaton M :



- Grammar G :

$$Q_0 \rightarrow a Q_0 \mid a Q_1$$

$$Q_1 \rightarrow b Q_1 \mid b Q_2$$

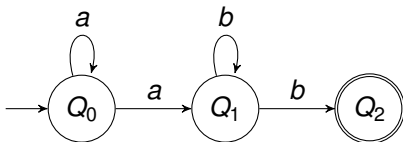
$$Q_2 \rightarrow \lambda$$

- Common word $aabb \in L(M) \cap L(G)$:

Computation	Derivation	Partial word
$[Q_0, aabb] \vdash [Q_0, abb]$	$Q_0 \Rightarrow a Q_0$	a
$\vdash [Q_1, bb]$	$\Rightarrow a a Q_1$	aa

Finite automaton to grammar: Example

- Automaton M :



- Grammar G :

$$Q_0 \rightarrow a Q_0 \mid a Q_1$$

$$Q_1 \rightarrow b Q_1 \mid b Q_2$$

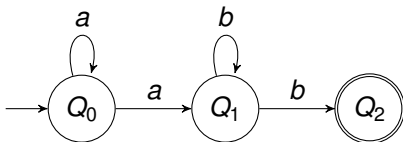
$$Q_2 \rightarrow \lambda$$

- Common word $aabb \in L(M) \cap L(G)$:

Computation	Derivation	Partial word
$[Q_0, aabb] \vdash [Q_0, abb]$	$Q_0 \Rightarrow a Q_0$	a
$\vdash [Q_1, bb]$	$\Rightarrow a a Q_1$	aa
$\vdash [Q_1, b]$	$\Rightarrow a a b Q_1$	aab

Finite automaton to grammar: Example

- Automaton M :



- Grammar G :

$$Q_0 \rightarrow a Q_0 \mid a Q_1$$

$$Q_1 \rightarrow b Q_1 \mid b Q_2$$

$$Q_2 \rightarrow \lambda$$

- Common word $aabb \in L(M) \cap L(G)$:

Computation	Derivation	Partial word
$[Q_0, aabb] \vdash [Q_0, abb]$	$Q_0 \Rightarrow a Q_0$	a
$\vdash [Q_1, bb]$	$\Rightarrow a a Q_1$	aa
$\vdash [Q_1, b]$	$\Rightarrow a a b Q_1$	aab
$\vdash [Q_2, \lambda]$	$\Rightarrow a a b b$	$aabb$

From finite automata to regular grammars

Theorem (Note: Material from §6.3!)

Given an NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$

Define $G = \langle Q, \Sigma, P, q_0 \rangle$ with

- $(q, a q') \in P$ for all $q \in Q$ and $q' \in \delta(q, a)$
- $(q, \lambda) \in P$ for all $q \in F$

Then $L(G) = L(M)$.

From finite automata to regular grammars

Theorem (Note: Material from §6.3!)

Given an NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$

Define $G = \langle Q, \Sigma, P, q_0 \rangle$ with

- $(q, a q') \in P$ for all $q \in Q$ and $q' \in \delta(q, a)$
- $(q, \lambda) \in P$ for all $q \in F$

Then $L(G) = L(M)$.

Intuition

- Every state in the automaton becomes a variable
- Every transition $q \xrightarrow{a} q'$ becomes a rule $q \rightarrow a q'$
- Every accepting state q becomes a rule $q \rightarrow \lambda$
- Paths in the automaton correspond to derivations
- Rules of the form $A \rightarrow a$ are not used

Regular grammar to automaton: Example

- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q =$
 $q_0 =$
 $F =$

Regular grammar to automaton: Example

- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q = \{S, B, Z\}$
 $q_0 =$
 $F =$

Regular grammar to automaton: Example

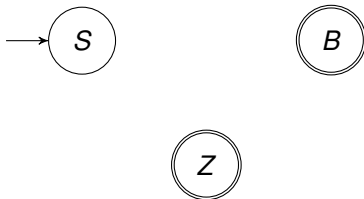
- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q = \{S, B, Z\}$
 $q_0 = S$
 $F =$

Regular grammar to automaton: Example

- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q = \{S, B, Z\}$
 $q_0 = S$
 $F = \{B, Z\}$

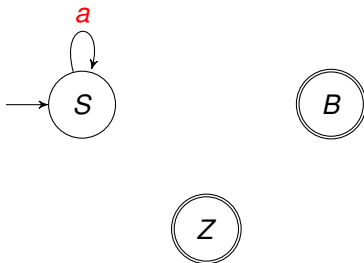
Regular grammar to automaton: Example

- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q = \{S, B, Z\}$
 $q_0 = S$
 $F = \{B, Z\}$



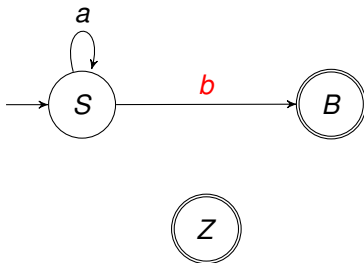
Regular grammar to automaton: Example

- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q = \{S, B, Z\}$
 $q_0 = S$
 $F = \{B, Z\}$



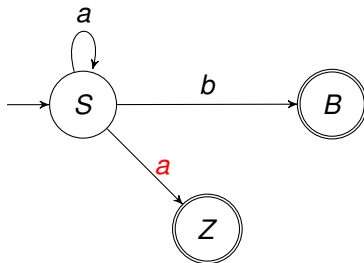
Regular grammar to automaton: Example

- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q = \{S, B, Z\}$
 $q_0 = S$
 $F = \{B, Z\}$



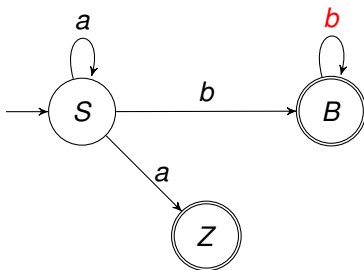
Regular grammar to automaton: Example

- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q = \{S, B, Z\}$
 $q_0 = S$
 $F = \{B, Z\}$



Regular grammar to automaton: Example

- Grammar: $S \rightarrow aS \mid bB \mid a$
 $B \rightarrow bB \mid \lambda$
- Automaton: $Q = \{S, B, Z\}$
 $q_0 = S$
 $F = \{B, Z\}$



From regular grammars to finite automata

Theorem

Given a regular CFG $G = \langle V, \Sigma, P, S \rangle$

Define $M = \langle V \cup \{Z\}, \Sigma, \delta, S, F \rangle$ with

- Z a special new variable, not in V
- $B \in \delta(A, a)$ for all $A \rightarrow aB$, and $Z \in \delta(A, a)$ for all $A \rightarrow a$
- $A \in F$ for all $A \rightarrow \lambda$, and $Z \in F$

Then $L(M) = L(G)$

From regular grammars to finite automata

Theorem

Given a regular CFG $G = \langle V, \Sigma, P, S \rangle$

Define $M = \langle V \cup \{Z\}, \Sigma, \delta, S, F \rangle$ with

- Z a special new variable, not in V
- $B \in \delta(A, a)$ for all $A \rightarrow aB$, and $Z \in \delta(A, a)$ for all $A \rightarrow a$
- $A \in F$ for all $A \rightarrow \lambda$, and $Z \in F$

Then $L(M) = L(G)$

Intuition

- Every variable of the grammar becomes a state
- Special (accepting) state z for terminating rules
- Rules $A \rightarrow aB$ become transitions $A \xrightarrow{a} B$
- Rules $A \rightarrow a$ become transitions $A \xrightarrow{a} Z$ (with $Z \in F$)
- Rules $A \rightarrow \lambda$ result in $A \in F$

Contents

- 1 Composition of Automata in Software Engineering
 - Product of Automata
 - Parallel composition
 - Projection and Hiding
- 2 Grammars and Derivations
 - Derivations
 - Derivation trees
 - Generated Language
- 3 Regular grammars
- 4 **Syntax of programming languages**
 - Ambiguity
 - Naive Parsing

Syntax of programming languages

- Completely modeled by **non-ambiguous context-free grammars**
- Example: CFG for syntax of expressions

$$\begin{aligned}
 \langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \langle \text{TermRest} \rangle_{opt} \\
 \langle \text{TermRest} \rangle &\rightarrow + \langle \text{Expr} \rangle \mid - \langle \text{Expr} \rangle \\
 \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \langle \text{FactorRest} \rangle_{opt} \\
 \langle \text{FactorRest} \rangle &\rightarrow * \langle \text{Term} \rangle \mid / \langle \text{Term} \rangle \\
 \langle \text{Factor} \rangle &\rightarrow \langle \text{Var} \rangle \mid \langle \text{Number} \rangle \mid (\langle \text{Expr} \rangle)
 \end{aligned}$$

Syntax of programming languages

- Completely modeled by **non-ambiguous context-free grammars**
- Example: CFG for syntax of expressions

$$\begin{aligned} \langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \langle \text{TermRest} \rangle_{opt} \\ \langle \text{TermRest} \rangle &\rightarrow + \langle \text{Expr} \rangle \mid - \langle \text{Expr} \rangle \\ \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \langle \text{FactorRest} \rangle_{opt} \\ \langle \text{FactorRest} \rangle &\rightarrow * \langle \text{Term} \rangle \mid / \langle \text{Term} \rangle \\ \langle \text{Factor} \rangle &\rightarrow \langle \text{Var} \rangle \mid \langle \text{Number} \rangle \mid (\langle \text{Expr} \rangle) \end{aligned}$$

- *Backus-Naur-Vorm* (BNF): notation-agreements
 - Non-terminals in angular brackets, for example $\langle \text{Number} \rangle$
 - Subscript $_{opt}$ (optional) for “ λ -choice:”
 - $\langle A \rangle \rightarrow \langle B \rangle_{opt}$ short for $\langle A \rangle \rightarrow \langle B \rangle \mid \lambda$
 - $\langle A \rangle \rightarrow \langle B \rangle_{opt} \langle C \rangle$ short for $\langle A \rangle \rightarrow \langle B \rangle \langle C \rangle \mid \langle C \rangle$
 - Many variations and extensions exist

Syntax of programming languages

- Completely modeled by **non-ambiguous context-free grammars**
- Example: CFG for syntax of expressions

$$\begin{aligned}
 \langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \langle \text{TermRest} \rangle_{opt} \\
 \langle \text{TermRest} \rangle &\rightarrow + \langle \text{Expr} \rangle \mid - \langle \text{Expr} \rangle \\
 \langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \langle \text{FactorRest} \rangle_{opt} \\
 \langle \text{FactorRest} \rangle &\rightarrow * \langle \text{Term} \rangle \mid / \langle \text{Term} \rangle \\
 \langle \text{Factor} \rangle &\rightarrow \langle \text{Var} \rangle \mid \langle \text{Number} \rangle \mid (\langle \text{Expr} \rangle)
 \end{aligned}$$

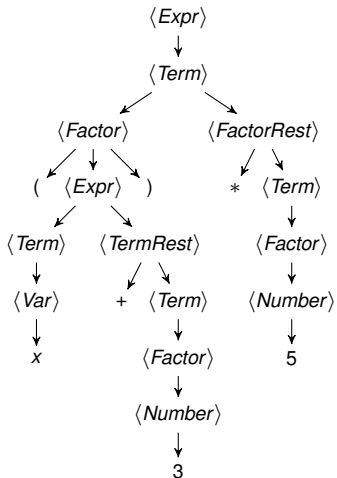
- *Backus-Naur-Vorm* (BNF): notation-agreements
 - Non-terminals in angular brackets, for example $\langle \text{Number} \rangle$
 - Subscript $_{opt}$ (optional) for “λ-choice:”
 - $\langle A \rangle \rightarrow \langle B \rangle_{opt}$ short for $\langle A \rangle \rightarrow \langle B \rangle \mid \lambda$
 - $\langle A \rangle \rightarrow \langle B \rangle_{opt} \langle C \rangle$ short for $\langle A \rangle \rightarrow \langle B \rangle \langle C \rangle \mid \langle C \rangle$
 - Many variations and extensions exist
- See book (appendix IV) for complete BNF grammar for Java programs

Syntax of programming languages: example

Parse (i.e., find a derivation for): $(x + 3) * 5$

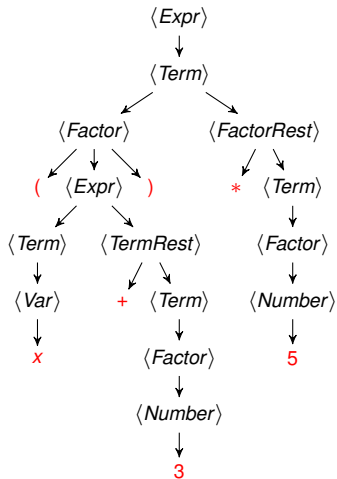
Syntax of programming languages: example

Parse (i.e., find a derivation for): $(x + 3) * 5$



Syntax of programming languages: example

Parse (i.e., find a derivation for): $(x + 3) * 5$



Example of ambiguity

- Sentence: *Look at the dog with one eye*

Example of ambiguity

- Sentence: *Look at the dog with one eye*
- Grammar:

Sentence → *VerbPhrase NounPhrase RelPhrase* |
VerbPhrase NounPhrase AdvPhrase

NounPhrase → *Pr. Det. PropoerNoun*

...

Example of ambiguity

- Sentence: *Look at the dog with one eye*

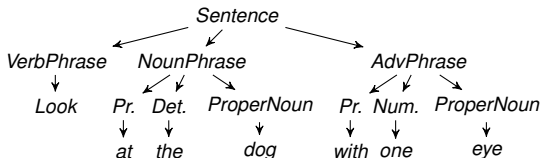
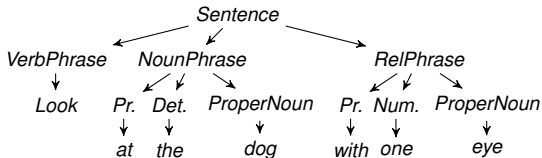
- Grammar:

Sentence → *VerbPhrase NounPhrase RelPhrase* |
VerbPhrase NounPhrase AdvPhrase

NounPhrase → *Pr. Det. ProperNoun*

...

- Derivation trees:



Example of ambiguity

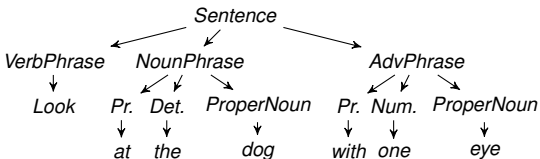
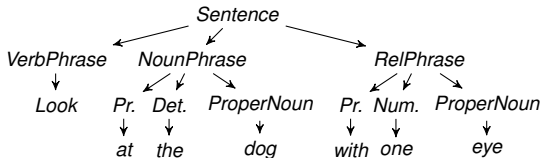
- Sentence: *Look at the dog with one eye*
- Grammar:

Sentence → *VerbPhrase NounPhrase RelPhrase* |
VerbPhrase NounPhrase AdvPhrase

NounPhrase → *Pr. Det. ProperNoun*

...

- Derivation trees:



Definition of (un)ambiguity

Definition

A CFG is *ambiguous* if a word exists in $L(G)$ with two distinct derivation trees.

Definition of (un)ambiguity

Definition

A CFG is *ambiguous* if a word exists in $L(G)$ with two distinct derivation trees.

Remarks

- Ambiguity results from the grammar, not from the language
The following language has an ambiguous and an unambiguous grammar:

$$G_1 : S \rightarrow aS \mid Sa \mid a \quad G_2 : S \rightarrow aS \mid a$$

$L(G_1) = L(G_2) = a^+$ holds; but G_1 is ambiguous, G_2 is not

Definition of (un)ambiguity

Definition

A CFG is *ambiguous* if a word exists in $L(G)$ with two distinct derivation trees.

Remarks

- Ambiguity results from the grammar, not from the language
The following language has an ambiguous and an unambiguous grammar:

$$G_1 : S \rightarrow aS \mid Sa \mid a \quad G_2 : S \rightarrow aS \mid a$$

$$L(G_1) = L(G_2) = a^+ \text{ holds; but } G_1 \text{ is ambiguous, } G_2 \text{ is not}$$
- **Fact:** There are also languages with *only* ambiguous grammars
Example: $L = \{a^m b^m c^n \mid m, n \geq 0\} \cup \{a^m b^n c^n \mid m, n \geq 0\}$

Definition of (un)ambiguity

Definition

A CFG is *ambiguous* if a word exists in $L(G)$ with two distinct derivation trees.

Remarks

- Ambiguity results from the grammar, not from the language
The following language has an ambiguous and an unambiguous grammar:

$$G_1 : S \rightarrow aS \mid Sa \mid a \quad G_2 : S \rightarrow aS \mid a$$

$$L(G_1) = L(G_2) = a^+ \text{ holds; but } G_1 \text{ is ambiguous, } G_2 \text{ is not}$$
- **Fact:** There are also languages with *only* ambiguous grammars
Example: $L = \{a^m b^m c^n \mid m, n \geq 0\} \cup \{a^m b^n c^n \mid m, n \geq 0\}$
- Unambiguous grammars exist for regular languages
- Programming languages have unambiguous grammars

The principle of parsing

Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

The principle of parsing

Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

Naive parsing algorithm: Search and backtrack

- 1 If the sentential form consists of only S , we are done
- 2 Search right side of a rule that matches part of the sentence

Success?

Failure?

The principle of parsing

Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

Naive parsing algorithm: Search and backtrack

- 1 If the sentential form consists of only S , we are done
- 2 Search right side of a rule that matches part of the sentence

Success?

- Replace the right side by the corresponding left side
- Repeat step 1 (next phase)

Failure?

The principle of parsing

Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

Naive parsing algorithm: Search and backtrack

- 1 If the sentential form consists of only S , we are done
- 2 Search right side of a rule that matches part of the sentence

Success?

- Replace the right side by the corresponding left side
- Repeat step 1 (next phase)

Failure?

- Go back to the previous phase
- Try the next possibility

The principle of parsing

Definition

- A *parsing* of a sentence is a derivation tree for that sentence
- A *parsing algorithm* for a grammar is an algorithm that produces a parsing for each sentence

Naive parsing algorithm: Search and backtrack

- 1 If the sentential form consists of only S , we are done
- 2 Search right side of a rule that matches part of the sentence

Success?

- Replace the right side by the corresponding left side
- Repeat step 1 (next phase)

Failure?

- Go back to the previous phase
- Try the next possibility

(backtracking)

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language:

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: *aaab*

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \quad \zeta$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \quad \zeta$
 $\leftarrow AaAb$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAb \leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAab \leftarrow$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAb \leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \end{matrix}$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAAb$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aaAb$

Example of naive parsing

• Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$

• Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$

• Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow ASb \downarrow$
 $\leftarrow aAab \leftarrow AAab \downarrow$
 $\leftarrow aAAb \leftarrow \dots \downarrow$
 $\leftarrow aaAb \leftarrow AaAb \downarrow$
 $\leftarrow aAAb \downarrow$
 $\leftarrow aAb$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA$ $A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$

		$\leftarrow AaAb$	$\leftarrow AAAb$	\downarrow
			$\leftarrow AAb$	\downarrow
			$\leftarrow ASb$	\downarrow
$\leftarrow aAab$	$\leftarrow AAab$	\downarrow		
	$\leftarrow aAAb$	$\leftarrow \dots$	\downarrow	
$\leftarrow aaAb$	$\leftarrow AaAb$	\downarrow		
	$\leftarrow aAAb$	\downarrow		
	$\leftarrow aAb$	$\leftarrow AAb$	\downarrow	

Example of naive parsing

• Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$

• Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$

• Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aaAb \leftarrow AaAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAb \leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow Ab \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$

Example of naive parsing

- Grammar: $S \rightarrow aSb \mid aA \quad A \rightarrow aA \mid a$
- Language: $\{a^m b^n \mid n \geq 0, m > n + 1\}$
- Parsing: $aaab \leftarrow Aaab \leftarrow AAab \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \\ \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AaAb \leftarrow AAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow ASb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAab \leftarrow AAab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAAb \leftarrow \dots \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aaAb \leftarrow AaAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aAb \leftarrow AAb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow Ab \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow Sb \begin{matrix} \downarrow \\ \downarrow \end{matrix}$
 $\leftarrow aSb \leftarrow S \quad \checkmark$

