Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson,
partly based on slides by Ana Bove

2024-01-15/16

Regular expressions

- Used in text editors:

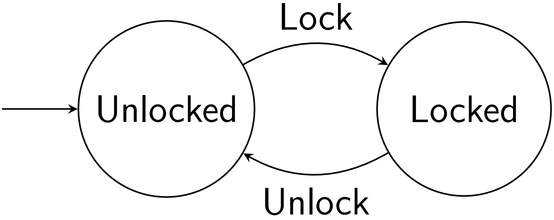
```
M-x replace-regexp RET
  add(\([^,]*\), \([^)]*\)) RET
  \1 + \2 RET
```

- Used to describe the lexical syntax of programming languages.
- Can only describe a limited class of "languages".

Finite automata

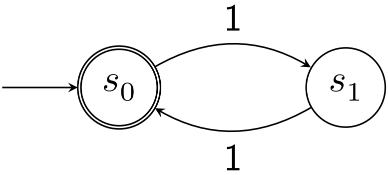
- Used to implement regular expression matching.
- Used to specify or model systems.
  - One kind of finite automaton is used in the specification of TCP.
- Equivalent to regular expressions.

Finite automata



Finite automata

Accepts strings of ones of even length:



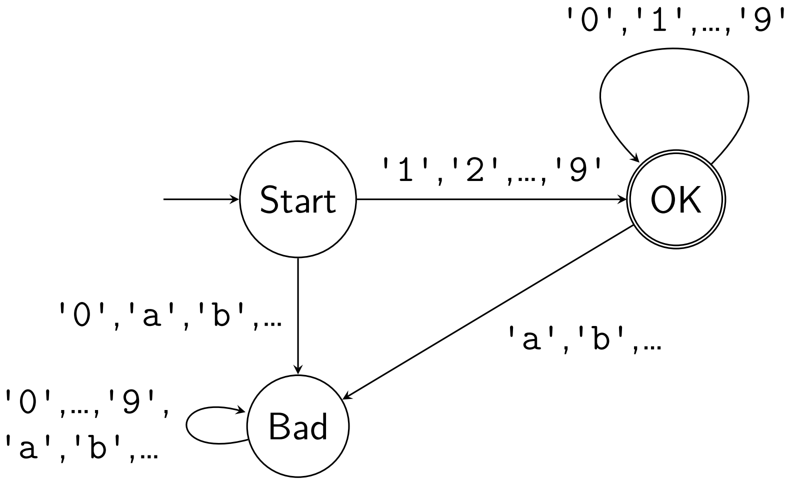
- The states are a kind of memory.
- Finite number of states $\Rightarrow$ finite memory.

Regular expressions

- A regular expression for strings of ones of even length: $(11)^*$.
- A regular expression for some keywords: $while \mid for \mid if \mid else$.
- A regular expression for positive natural number literals (of a certain form): $[1\text{–}9][0\text{–}9]^*$.

Finite automata

Accepts positive natural number literals:



Conversions

- We will see how to convert between regular expressions and finite automata.
- In fact, we will discuss several kinds of finite automata, and conversions between the different kinds.

Context-free grammars

- More general than regular expressions.
- Used to describe the syntax of programming languages.
- Used by parser generators. (Often restricted.)

Context-free grammars

$$
\begin{aligned}
\mathit{Expr} &::= \mathit{Number} \\
&\quad|\quad \mathit{Expr}\ \mathit{Op}\ \mathit{Expr} \\
&\quad|\quad \texttt{'('}\ \mathit{Expr}\ \texttt{')'} \\
\mathit{Op} &::= \texttt{'+'} \mid \texttt{'-'} \mid \texttt{'*'} \mid \texttt{'/'}
\end{aligned}
$$

Turing machines

- A model of what it means to “compute”:
  - Unbounded memory: an infinite tape of cells.
  - A read/write head that can move along the tape.
  - A kind of finite state machine with rules for what the head should do.
- Equivalent to a number of other models of computation.

Proofs

- Used to make it more likely that arguments are correct.
- Used to make arguments more convincing.

Induction

- Regular induction for $\mathbb{N}$.
- Complete (strong, course of values) induction for $\mathbb{N}$.

Inductively defined sets

- An example:
  The natural numbers ($\mathbb{N} = \{ 0, 1, 2, ... \}$).
- Structural induction for
  inductively defined sets.

General information

See the course web pages.

Repetition (?) of some classical logic

Propositions

- A proposition is, roughly speaking, some statement that is true or false.
  - $2 = 3$.
  - The program `while true do {x := 4}` terminates.
  - $P = NP$.
  - If $P = NP$, then $2 = 3$.
- It may not always be known what the truth value ($\top$ or $\bot$) of a proposition is.

Some logical connectives

- And: $\wedge$.
- Or: $\vee$.
- Not: $\neg$.
- Implies: $\Rightarrow$.
- If and only if (iff): $\Leftrightarrow$.

Some logical connectives

Truth tables for these connectives:

p	q	$p \wedge q$	$p \vee q$	$\neg p$	$p \Rightarrow q$	$p \Leftrightarrow q$
$\top$	$\top$	$\top$	$\top$	$\perp$	$\top$	$\top$
$\top$	$\perp$	$\perp$	$\top$		$\perp$	$\perp$
$\perp$	$\top$	$\perp$	$\top$	$\top$	$\top$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$		$\top$	$\top$

Note that $p \Rightarrow q$ is true if p is false.

Lecture quizzes

- I will ask you questions during the lectures.
- You can reply anonymously via something called Pingo.
- First you get to discuss the answers with other students.

Which of the following truth tables are correct for the proposition $(p \vee q) \Rightarrow p$?

A:

p	q	$(p \vee q) \Rightarrow p$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$

B:

p	q	$(p \vee q) \Rightarrow p$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\bot$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$

C:

p	q	$(p \vee q) \Rightarrow p$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\bot$	$\top$	$\bot$
$\bot$	$\bot$	$\top$

D:

p	q	$(p \vee q) \Rightarrow p$
$\top$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\top$

Respond at `https://pingo.coactum.de/729558`.

Validity

- A proposition is *valid*, or a *tautology*, if it is satisfied for all assignments of truth values to its variables.
- Examples:
  - $p \Rightarrow p$.
  - $p \vee \neg p$.

Logical equivalence

- Two propositions p and q are *logically equivalent* if they have the same truth tables, i.e. if $p \Leftrightarrow q$ is valid.
- Examples:
  - $\neg\neg p \quad \Leftrightarrow \quad p$.
  - $(p \Leftrightarrow q) \quad \Leftrightarrow \quad (p \Rightarrow q) \wedge (q \Rightarrow p)$.
  - $p \wedge q \quad \Leftrightarrow \quad q \wedge p$.
  - $p \wedge (q \vee r) \quad \Leftrightarrow \quad (p \wedge q) \vee (p \wedge r)$.
  - $p \wedge (p \vee q) \quad \Leftrightarrow \quad p$.

Which of the following propositions are valid?

1. $(p \Rightarrow q) \Leftrightarrow \neg p \vee q$.
2. $(p \Rightarrow q) \Leftrightarrow p \vee \neg q$.
3. $\neg(p \wedge q) \Leftrightarrow \neg p \wedge \neg q$.
4. $\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$.
5. $((p \Rightarrow p) \Rightarrow q) \Rightarrow p$.
6. $((p \Rightarrow q) \Rightarrow p) \Rightarrow p$.

Respond at `https://pingo.coactum.de/729558`.

Predicates

A predicate is, roughly speaking, a function to propositions.

- $P(n) =$ "n is a prime number".
- $Q(a, b) =$ "$(a + b)^2 = a^2 + 2ab + b^2$".

Quantifiers

Quantifiers:

- For all: $\forall$.
  - $\forall x.\ x = x$.
  - $\forall a, b \in \mathbb{R}.\ (a + b)^2 = a^2 + 2ab + b^2$.
- There exists: $\exists$.
  - $\exists n \in \mathbb{N}.\ n = 2n$.

Which of the following propositions, involving predicate variables, are valid?

1. $(\neg\forall n \in \mathbb{N}.\ P(n)) \Leftrightarrow (\forall n \in \mathbb{N}.\ \neg P(n))$.
2. $(\neg\forall n \in \mathbb{N}.\ P(n)) \Leftrightarrow (\exists n \in \mathbb{N}.\ \neg P(n))$.
3. $(\forall m \in \mathbb{N}.\ \exists n \in \mathbb{N}.\ P(m, n)) \Leftrightarrow$
   $(\exists n \in \mathbb{N}.\ \forall m \in \mathbb{N}.\ P(m, n))$.

Respond at `https://pingo.coactum.de/729558`.

Repetition (?) of some set theory

Sets

- A *set* is, roughly speaking, a collection of elements.
- Some notation for defining sets:
  - $\{ 0, 1, 2, 4, 8 \}$.
  - $\{ n \in \mathbb{N} \mid n > 2 \}$.
  - $\{ 2^n \mid n \in \mathbb{N} \}$.

Members, subsets

- Membership: $\in$.
  - $4 \in \{ 2^n \mid n \in \mathbb{N} \}$.
  - $2 \notin \{ n \in \mathbb{N} \mid n > 2 \}$.
- Two sets are equal if they have the same elements: $(A = B) \Leftrightarrow (\forall x.\ x \in A \Leftrightarrow x \in B)$.
- Subset relation: $(A \subseteq B) \Leftrightarrow (\forall x.\ x \in A \Rightarrow x \in B)$.
  - $\{ 2^n \mid n \in \mathbb{N} \} \subseteq \mathbb{N}$.
  - $\{ 0, 1, 2, 4, 8 \} \not\subseteq \{ n \in \mathbb{N} \mid n > 2 \}$.

An aside

- Unrestricted naive set theory can be inconsistent.
- Russell's paradox:
  - Define $S = \{ X \mid X \notin X \}$, where X ranges over all sets.
  - We have $S \in S \Leftrightarrow S \notin S$!
  - One can fix this problem by imposing rules that ensure that S is not a set.

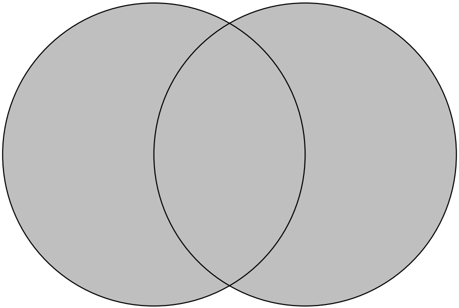
Set operations

The empty set: $\emptyset$.

Set operations

Union:

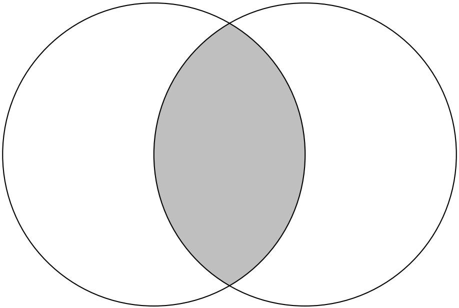
$$A \cup B = \{ \, x \mid x \in A \vee x \in B \, \} \, .$$



Set operations

Intersection:

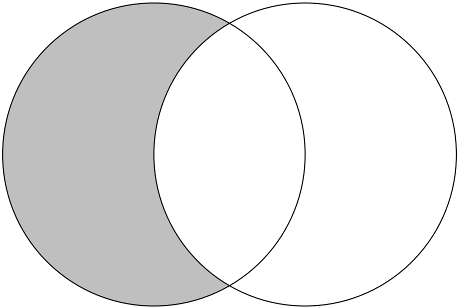
$$A \cap B = \{ \, x \mid x \in A \wedge x \in B \, \} \, .$$



Set operations

Set difference:

$$A \setminus B = A - B = \{\, x \in A \mid x \notin B \,\}.$$

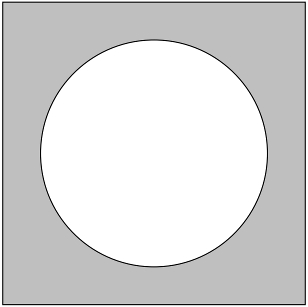


Set operations

Complement:

$$\overline{A} = U \setminus A$$

(if U is fixed in advance and $A \subseteq U$).



Set operations

Cartesian product:

$$A \times B = \{\, (x, y) \mid x \in A \wedge y \in B \,\}.$$

$$\begin{aligned} &\{\, a, b \,\} \times \{\, 0, 1 \,\} = \\ &\quad \{\, (a, 0), (a, 1), (b, 0), (b, 1) \,\} \end{aligned}$$

Set operations

Power set:

$$\wp(S) = 2^S = \{\, A \mid A \subseteq S \,\}\,.$$

$$\begin{aligned}
&\wp(\{\, 0, 1, 2 \,\}) = \\
&\quad \{\emptyset, \\
&\qquad \{\, 0 \,\}, \{\, 1 \,\}, \{\, 2 \,\}, \\
&\qquad \{\, 0, 1 \,\}, \{\, 0, 2 \,\}, \{\, 1, 2 \,\}, \\
&\qquad \{\, 0, 1, 2 \,\}\}
\end{aligned}$$

Set operations

The set of all finite subsets of a set:

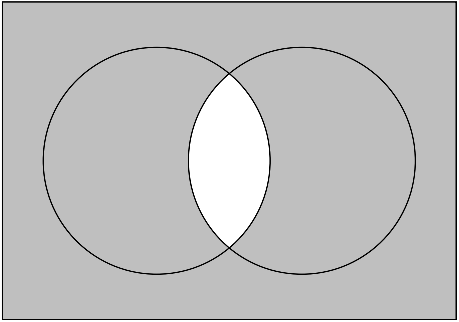
$$\mathrm{Fin}(S) = \{\, A \mid A \subseteq S, A \text{ is finite} \,\}\,.$$

Which of the following propositions are valid? Variables range over sets. U is non-empty.

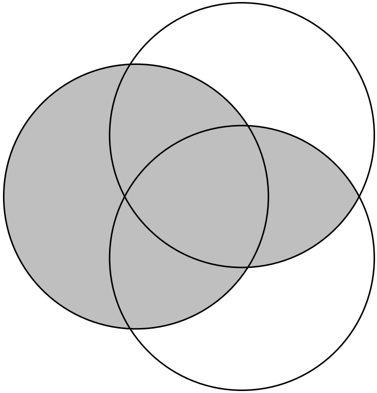
1. $\overline{A \cap B} = \overline{A} \cap \overline{B}$.
2. $\overline{A \cap B} = \overline{A} \cup \overline{B}$.
3. $\emptyset = \{ \emptyset \}$.
4. $A \in \wp(A)$.
5. $A \cup (B \cap C) = (A \cup B) \cap C$.
6. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

Respond at `https://pingo.coactum.de/729558`.

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$



$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$



Relations

- A binary relation R on A is a subset of $A^2 = A \times A$: $R \subseteq A^2$.
- Notation: xRy means the same as $(x, y) \in R$.
- Can be generalised from $A \times A$ to $A \times B \times C \times \cdots$.

Some binary relation properties

For $R \subseteq A \times B$:

- Total (left-total): $\forall x \in A.\ \exists y \in B.\ xRy$.
- Functional/deterministic:
  $\forall x \in A.\ \forall y, z \in B.\ xRy \wedge xRz \Rightarrow y = z$.

Functions

- The set of *functions* from the set A to the set B is denoted by $A \to B$.
- It is sometimes defined as the set of total and functional relations $f \subseteq A \times B$.
- Notation: $f(x) = y$ means $(x, y) \in f$.
- If the requirement of totality is dropped, then we get the set of *partial* functions, $A \rightharpoonup B$.
- The *domain* is A, and the *codomain* B.
- The *image* is $\{ y \in B \mid x \in A, f(x) = y \}$.

Which of the following relations on $\{ a, b \}$ are functions?

1. $\{ \}$.
2. $\{ (a, a) \}$.
3. $\{ (a, a), (a, b) \}$.
4. $\{ (a, a), (b, a) \}$.
5. $\{ (a, a), (b, a), (b, b) \}$.

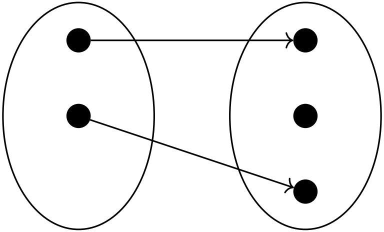
Respond at `https://pingo.coactum.de/729558`.

Identity, composition

- The *identity function* id on a set A is defined by $id(x) = x$.
- For functions $f \in B \to C$ and $g \in A \to B$ the *composition* $f \circ g \in A \to C$ is defined by $(f \circ g)(x) = f(g(x))$.

Injections

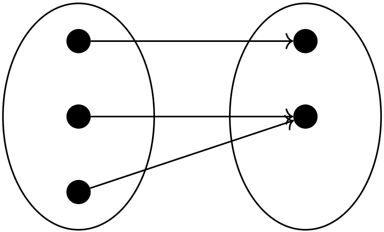
The function $f \in A \to B$ is *injective* if
$\forall x, y \in A.\ f(x) = f(y) \Rightarrow x = y$.



- Every input is mapped to a unique output.
- A is "no larger than" B.
- Holds if f has a left inverse $g \in B \to A$: $g \circ f = id$.

Surjections

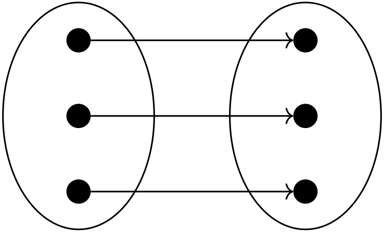
The function $f \in A \to B$ is *surjective* if $\forall y \in B.\ \exists x \in A.\ f(x) = y$.



- The function "targets" every element in the codomain.
- A is "no smaller than" B.
- Holds if f has a right inverse $g \in B \to A$: $f \circ g = id$.

Bijections

The function $f \in A \to B$ is *bijective* if it is both injective and surjective.



- A and B have the same “size”.
- Holds if and only if f has a left and right inverse $g \in B \to A$.

Which of the following functions are injective? Surjective?

- $f \in \mathbb{N} \to \mathbb{N}$, $f(n) = n + 1$.
- $g \in \mathbb{Z} \to \mathbb{Z}$, $g(i) = i + 1$.
- $h \in \mathbb{N} \to Bool$, $h(n) = \begin{cases} \text{true}, & \text{if } n \text{ is even}, \\ \text{false}, & \text{otherwise.} \end{cases}$

Respond at `https://pingo.coactum.de/729558`.

The pigeonhole principle

- If there are n pigeonholes, and $m > n$ pigeons in these pigeonholes, then at least one pigeonhole must contain more than one pigeon.
- If $f \in \{ k \in \mathbb{N} \mid k < m \} \to \{ k \in \mathbb{N} \mid k < n \}$ for $m, n \in \mathbb{N}$, and $m > n$, then f is not injective.

More binary relation properties

For $R \subseteq A^2$:

- Reflexive: $\forall x \in A.\ xRx$.
- Symmetric: $\forall x, y \in A.\ xRy \Rightarrow yRx$.
- Transitive: $\forall x, y, z \in A.\ xRy \wedge yRz \Rightarrow xRz$.
- Antisymmetric:
  $\forall x, y \in A.\ xRy \wedge yRx \Rightarrow x = y$.

Partial orders

A *partial order* is reflexive, antisymmetric and transitive.

- $\leq$ for $\mathbb{N}$.
- Not $<$.

Which of the following sets are partial orders on $\{ 0, 1 \}$?

1. $\{ (0, 0) \}$.
2. $\{ (0, 0), (1, 1) \}$.
3. $\{ (0, 0), (0, 1), (1, 1) \}$.
4. $\{ (0, 0), (0, 1), (1, 0) \}$.

Respond at `https://pingo.coactum.de/729558`.

Equivalence relations

An *equivalence relation* is reflexive, symmetric and transitive.

- $\{ (n,n) \mid n \in \mathbb{N} \} \subseteq \mathbb{N}^2$.
- Not $\{ (n,n) \mid n \in \mathbb{N} \} \subseteq \mathbb{R}^2$.

Which of the following sets are equivalence relations on $\{ 0, 1 \}$?

1. $\{ (0, 0) \}$.
2. $\{ (0, 0), (1, 1) \}$.
3. $\{ (0, 0), (0, 1), (1, 0) \}$.
4. $\{ (0, 0), (0, 1), (1, 0), (1, 1) \}$.

Respond at `https://pingo.coactum.de/729558`.

Partitions

A partition of the set A is a set $P \subseteq \wp(A)$ satisfying the following properties:

- Every element is non-empty: $\forall B \in P.\ B \neq \emptyset$.
- The elements cover A: $\bigcup_{B \in P} B = A$.
- The elements are mutually disjoint: $\forall B, C \in P.\ B \neq C \ \Rightarrow\ B \cap C = \emptyset$.

Partitions

Example:

$$\{\,\{\,1,2\,\}\,,\{\,3,5\,\}\,,\{\,4\,\}\,\}$$

is a partition of

$$\{\,1,2,3,4,5\,\}\,.$$

Equivalence classes

- The equivalence classes of an equivalence relation R on A: $[x]_R = \{ y \in A \mid xRy \}$.
- Note that $\forall x, y \in A.\ [x]_R = [y]_R \Leftrightarrow xRy$. Proof sketch:
  - $\Rightarrow$: Assume $[x]_R = [y]_R$. We have yRy, so $y \in [y]_R$, $y \in [x]_R$, and xRy.
  - $\Leftarrow$: Assume xRy.
    - $[x]_R \subseteq [y]_R$: If $z \in [x]_R$, then xRz, so yRz, and thus $z \in [y]_R$.
    - $[y]_R \subseteq [x]_R$: Similar.

Equivalence classes

- The equivalence classes of an equivalence relation R on A: $[x]_R = \{ y \in A \mid xRy \}$.
- The set of equivalence classes $\{ [x]_R \mid x \in A \}$ partitions A. Proof sketch:
  - $[x]_R \neq \emptyset$ because $x \in [x]_R$.
  - $\bigcup_{B \in \{ [x]_R \mid x \in A \}} B = \bigcup_{x \in A} [x]_R = A$.
  - Assume that $z \in [x]_R \cap [y]_R$. We get that xRz and yRz, so we have xRy and thus $[x]_R = [y]_R$.

Equivalence classes

- The equivalence classes of an equivalence relation R on A: $[x]_R = \{ y \in A \mid xRy \}$.
- The quotient set $A/R = \{ [x]_R \mid x \in A \}$.

Quotients

- Can one define $\mathbb{Z} = \mathbb{N}^2$, with the intention that (m, n) stands for $m - n$?
- No, $(0, 1)$ and $(1, 2)$ would both represent -1.
- Instead one can use a quotient set:

$$\mathbb{Z} = \mathbb{N}^2 / \sim_{\mathbb{Z}} \, ,$$

where

$$(m_1, n_1) \sim_{\mathbb{Z}} (m_2, n_2) \Leftrightarrow m_1 + n_2 = m_2 + n_1.$$

Quotients

Another example:

$$\mathbb{Q} = \{ \, (m, n) \mid m \in \mathbb{Z}, n \in \mathbb{N} \setminus \{ \, 0 \, \} \, \} \, / \sim_{\mathbb{Q}} \, ,$$

where

$$(m_1, n_1) \sim_{\mathbb{Q}} (m_2, n_2) \Leftrightarrow m_1 n_2 = m_2 n_1 .$$

Functions from quotients

- Sometimes you see functions defined in the following way:

$$\begin{aligned} & f \in A/{\sim} \rightarrow B \\ & f([x]) = g(x) \end{aligned}$$

- If $x \sim y$, then $[x] = [y]$, so we should have $f([x]) = f([y])$.
- This follows if $x \sim y$ implies that $g(x) = g(y)$.

Functions from quotients

- An example:

$$
\begin{aligned}
&-\_ \in \mathbb{Z} \to \mathbb{Z} \\
&-[(m, n)] = [(n, m)]
\end{aligned}
$$

- Take $p_1 = (m_1, n_1)$ and $p_2 = (m_2, n_2)$.
- If $p_1 \sim_{\mathbb{Z}} p_2$, i.e. if $(m_1, n_1) \sim_{\mathbb{Z}} (m_2, n_2)$, then $(n_1, m_1) \sim_{\mathbb{Z}} (n_2, m_2)$, and thus $-[p_1] = -[p_2]$.

Which of the following propositions are true?

1. $[(2,5)]_{\sim_{\mathbb{Z}}} = [(0,3)]_{\sim_{\mathbb{Z}}}$.
2. $[(2,5)]_{\sim_{\mathbb{Z}}} = [(3,0)]_{\sim_{\mathbb{Z}}}$.
3. $[(2,5)]_{\sim_{\mathbb{Q}}} = [(4,10)]_{\sim_{\mathbb{Q}}}$.
4. $[(2,5)]_{\sim_{\mathbb{Q}}} = [(10,4)]_{\sim_{\mathbb{Q}}}$.

Respond at `https://pingo.coactum.de/729558`.

Next lecture

- Proofs.
- Induction for the natural numbers.
- Inductively defined sets.
- Recursive functions.

Deadline for the first quiz: 2024-01-18, 13:00.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson,
partly based on slides by Ana Bove

2024-01-18

Today

- Proofs.
- Induction for the natural numbers.
- Inductively defined sets.
- Recursive functions.

Some basic proof methods

Some basic proof methods

- To prove $p \Rightarrow q$, assume p and prove q.
- To prove $\forall x \in A.\ P(x)$, assume that we have an $x \in A$ and prove $P(x)$.
- To prove $p \Leftrightarrow q$, prove both $p \Rightarrow q$ and $q \Rightarrow p$.
- To prove $\neg p$, assume p and derive a contradiction.
- To prove p, prove $\neg\neg p$.
- To prove $p \Rightarrow q$, assume $\neg q$ and prove $\neg p$.

(There may be other ways to prove these things.)

Induction

Mathematical induction

For a natural number predicate P we can prove $\forall n \in \mathbb{N}.\ P(n)$ in the following way:

- Prove $P(0)$.
- For every $n \in \mathbb{N}$, prove that $P(n)$ implies $P(n+1)$.

With a formula:

$$P(0) \wedge (\forall n \in \mathbb{N}.\ P(n) \Rightarrow P(n+1)) \Rightarrow \\ \forall n \in \mathbb{N}.\ P(n)$$

Which of the following variants of induction are valid?

1. $P(0) \wedge (\forall n \in \mathbb{N}.\ n \geq 1 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N}.\ n \geq 1 \Rightarrow P(n)$.
2. $P(1) \wedge (\forall n \in \mathbb{N}.\ n \geq 1 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N}.\ n \geq 1 \Rightarrow P(n)$.
3. $P(1) \wedge P(2) \wedge (\forall n \in \mathbb{N}.\ n \geq 2 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N}.\ n \geq 1 \Rightarrow P(n)$.

Respond at `https://pingo.coactum.de/729558`.

Counterexamples

- One can sometimes prove that a statement is invalid by using a counterexample.
- Example: The following statement does not hold for $P(n) := n \neq 1$ and $n = 1$:

$$P(0) \wedge (\forall n \in \mathbb{N}.\ n \geq 1 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N}.\ n \geq 1 \Rightarrow P(n)$$

The hypotheses hold, but not the conclusion.

Counterexamples

More carefully:

- Let us prove

$$\neg(\forall \text{ natural number predicates } P.\ P(0) \wedge (\forall n \in \mathbb{N}.\ n \geq 1 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N}.\ n \geq 1 \Rightarrow P(n)).$$

- We assume

$$\forall \text{ natural number predicates } P.\ P(0) \wedge (\forall n \in \mathbb{N}.\ n \geq 1 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N}.\ n \geq 1 \Rightarrow P(n),$$

and derive a contradiction.

Counterexamples

- Let us use the predicate $P(n) := n \neq 1$.
- We have $P(0)$, i.e. $0 \neq 1$.
- We also have
  $\forall n \in \mathbb{N}.\ n \geq 1 \wedge P(n) \Rightarrow P(n+1)$, i.e.
  $\forall n \in \mathbb{N}.\ n \geq 1 \wedge n \neq 1 \Rightarrow n+1 \neq 1$.
- Thus we get $\forall n \in \mathbb{N}.\ n \geq 1 \Rightarrow P(n)$.
- Let us use $n = 1$.
- We have $1 \geq 1$.
- Thus we get $P(1)$, i.e. $1 \neq 1$.
- This is a contradiction, so we are done.

Complete induction

We can also prove $\forall n \in \mathbb{N}.\ P(n)$ in the following way:

- Prove $P(0)$.
- For every $n \in \mathbb{N}$, prove that if $P(i)$ holds for every natural number $i \leq n$, then $P(n+1)$ holds.

With a formula:

$$\begin{aligned}
&P(0) \wedge \\
&(\forall n \in \mathbb{N}.\ (\forall i \in \mathbb{N}.\ i \leq n \Rightarrow P(i)) \Rightarrow P(n+1)) \Rightarrow \\
&\quad \forall n \in \mathbb{N}.\ P(n)
\end{aligned}$$

Which of the following variants of complete induction are valid?

1. $(\forall n \in \mathbb{N}.\ (\forall i \in \mathbb{N}.\ i < n \Rightarrow P(i)) \Rightarrow P(n)) \Rightarrow \forall n \in \mathbb{N}.\ P(n)$.

2. $P(1) \wedge$
$$(\forall n \in \mathbb{N}.\ n \geq 1 \wedge (\forall i \in \mathbb{N}.\ i \leq n \Rightarrow P(i)) \Rightarrow P(n+1)) \Rightarrow$$
$\forall n \in \mathbb{N}.\ P(n)$.

Respond at `https://pingo.coactum.de/729558`.

An example

Lemma

Every natural number $n \geq 8$ can be written as a sum of multiples of 3 and 5.

An example

Proof.

Let $P(n)$ be $n \geq 8 \Rightarrow \exists i, j \in \mathbb{N}.\ n = 3i + 5j$. We prove that $P(n)$ holds for all $n \in \mathbb{N}$ by complete induction on n:

- Base cases ($n = 0, ..., 7$): Trivial.
- Base cases ($n = 8$, $n = 9$, $n = 10$): Easy.
- Step case ($n \geq 10$, inductive hypothesis $\forall i \in \mathbb{N}.\ i \leq n \Rightarrow P(i)$, goal $P(n+1)$): Because $n - 2 \geq 8$ the inductive hypothesis for $n - 2$ implies that there are $i, j \in \mathbb{N}$ such that $n - 2 = 3i + 5j$. Thus we get $1 + n = 3 + (n - 2) = 3(i + 1) + 5j$. $\square$

Proofs

How detailed should a proof be?

- Depends on the purpose of the proof.
- Who or what do you want to convince?
  - Yourself?
  - A fellow student?
  - An examiner?
  - An experienced researcher?
  - A computer program (a proof checker)?

Discuss the following proof of $\forall n \in \mathbb{N}.\ \sum_{i=0}^{n} i = n\frac{n+1}{2}$. Would you like to add/remove/change anything?

By induction on n:

- $n = 0$: $\sum_{i=0}^{0} i = 0 = 0\frac{0+1}{2}$.
- $n = k + 1$, $k \in \mathbb{N}$:

$$\sum_{i=0}^{n} i = \sum_{i=0}^{k+1} i = (k+1) + \sum_{i=0}^{k} i =$$

$$(k+1) + k\frac{k+1}{2} =$$

$$(k+1)\left(1 + \frac{k}{2}\right) = (k+1)\frac{k+2}{2}.$$

Respond at `https://pingo.coactum.de/729558`.

Inductively defined sets

Inductively defined sets

The natural numbers:

$$\frac{}{\mathsf{zero} \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{\mathsf{suc}(n) \in \mathbb{N}}$$

Compare:

```
data Nat = Zero | Suc Nat
```

Inductively defined sets

Booleans:

$$\frac{}{\mathsf{true} \in Bool} \qquad \frac{}{\mathsf{false} \in Bool}$$

Compare:

```
data Bool = True | False
```

Inductively defined sets

Finite lists:

$$\frac{}{\mathsf{nil} \in List(A)} \qquad \frac{x \in A \qquad xs \in List(A)}{\mathsf{cons}(x, xs) \in List(A)}$$

Compare:

```
data List a = Nil | Cons a (List a)
```

Which of the following expressions are lists of natural numbers (members of $List(\mathbb{N})$)?

1. nil.
2. $\mathsf{cons}(\mathsf{nil}, 5)$.
3. $\mathsf{cons}(5, \mathsf{nil})$.

Respond at `https://pingo.coactum.de/729558`.

Lists

Alternative notation for lists:

- $[\,]$ instead of nil.
- $x : xs$ instead of $\text{cons}(x, xs)$.
- $[1, 2, 3]$ instead of $\text{cons}(1, \text{cons}(2, \text{cons}(3, \text{nil})))$.

Recursive functions

Recursive functions

An example:

$$
\begin{aligned}
&length \in List(A) \to \mathbb{N} \\
&length(\mathsf{nil}) &&= \mathsf{zero} \\
&length(\mathsf{cons}(x, xs)) &&= \mathsf{suc}(length(xs))
\end{aligned}
$$

Recursive functions

$$\begin{aligned}
&length([1,2,3]) &=\\
&length(\mathsf{cons}(1,\mathsf{cons}(2,\mathsf{cons}(3,\mathsf{nil})))) &=\\
&\mathsf{suc}(length(\mathsf{cons}(2,\mathsf{cons}(3,\mathsf{nil})))) &=\\
&\mathsf{suc}(\mathsf{suc}(length(\mathsf{cons}(3,\mathsf{nil})))) &=\\
&\mathsf{suc}(\mathsf{suc}(\mathsf{suc}(length(\mathsf{nil})))) &=\\
&\mathsf{suc}(\mathsf{suc}(\mathsf{suc}(\mathsf{zero}))) &=\\
&3
\end{aligned}$$

Recursive functions

Not well-defined:

$$\begin{aligned}
&bad \in List(A) \to \mathbb{N} \\
&bad(\mathsf{nil}) &&= \mathsf{zero} \\
&bad(\mathsf{cons}(x, xs)) &&= bad(\mathsf{cons}(x, xs))
\end{aligned}$$

Recursive functions

Another example:

$$
\begin{aligned}
& f \in List(A) \times List(A) \to List(A) \\
& f(\mathsf{nil}, \phantom{\mathsf{cons}(x, xs),}\; ys) = ys \\
& f(\mathsf{cons}(x, xs), ys) = \mathsf{cons}(x, f(xs, ys))
\end{aligned}
$$

What is the result of $f([1, 2], [3, 4])$?

1. $[1, 2, 3, 4]$.
2. $[4, 3, 2, 1]$.
3. $[2, 1, 4, 3]$.
4. $[1, 3, 2, 4]$.
5. $[1, 4, 2, 3]$.

Respond at `https://pingo.coactum.de/729558`.

Recursive functions

$$
\begin{aligned}
&append \in List(A) \times List(A) \to List(A) \\
&append(\mathsf{nil}, \qquad\qquad\quad ys) = ys \\
&append(\mathsf{cons}(x, xs), ys) = \mathsf{cons}(x, append(xs, ys))
\end{aligned}
$$

Mutual induction

Mutual induction

- Two mutually defined functions:

$$
\begin{aligned}
&odd, even \in \mathbb{N} \to Bool \\
&odd(\mathsf{zero}) &&= \mathsf{false} \\
&odd(\mathsf{suc}(n)) &&= even(n) \\
&even(\mathsf{zero}) &&= \mathsf{true} \\
&even(\mathsf{suc}(n)) &&= odd(n)
\end{aligned}
$$

- Another function:

$$
\begin{aligned}
&odd' \in \mathbb{N} \to Bool \\
&odd'(\mathsf{zero}) &&= \mathsf{false} \\
&odd'(\mathsf{suc}(n)) &&= not(odd'(n))
\end{aligned}
$$

- Can we prove $\forall n \in \mathbb{N}. odd(n) = odd'(n)$?

Mutual induction

First attempt:

- Let us use mathematical induction.
- Inductive hypothesis:

$$P(n) := odd(n) = odd'(n)$$

- Base case (P(zero)):

$$\begin{array}{ll} odd(\text{zero}) & = \\ \text{false} & = \\ odd'(\text{zero}) & \end{array}$$

Mutual induction

Step case ($\forall n \in \mathbb{N}. P(n) \Rightarrow P(\mathsf{suc}(n))$):

- Given $n \in \mathbb{N}$, let us assume $odd(n) = odd'(n)$:

$$\begin{aligned}
&odd(\mathsf{suc}(n)) &&= \\
&even(n) &&= \{???\} \\
&not(odd'(n)) &&= \\
&odd'(\mathsf{suc}(n)).
\end{aligned}$$

Mutual induction

Step case ($\forall n \in \mathbb{N}. P(n) \Rightarrow P(\mathsf{suc}(n))$):

- Given $n \in \mathbb{N}$, let us assume $odd(n) = odd'(n)$:

$$\begin{aligned} & odd(\mathsf{suc}(n)) && = \\ & even(n) && = \{???\} \\ & not(odd'(n)) && = \\ & odd'(\mathsf{suc}(n)). && \end{aligned}$$

- Let us generalise the inductive hypothesis:

$$\begin{aligned} P(n) := {} & odd(n) = odd'(n) \wedge \\ & even(n) = not(odd'(n)) \end{aligned}$$

Mutual induction

Base case ($P(\text{zero})$):

- First part:

$$\begin{aligned} odd(\text{zero}) &= \\ \text{false} &= \\ odd'(\text{zero}) \end{aligned}$$

- Second part:

$$\begin{aligned} even(\text{zero}) &= \\ \text{true} &= \\ not(\text{false}) &= \\ not(odd'(\text{zero})) \end{aligned}$$

Mutual induction

Step case ($\forall n \in \mathbb{N}. P(n) \Rightarrow P(\mathsf{suc}(n))$):

- Given $n \in \mathbb{N}$, let us assume $odd(n) = odd'(n)$ and $even(n) = not(odd'(n))$.
- First part:

$$\begin{array}{ll} odd(\mathsf{suc}(n)) & = \\ even(n) & = \{\text{By the second IH.}\} \\ not(odd'(n)) & = \\ odd'(\mathsf{suc}(n)) & \end{array}$$

Mutual induction

Step case ($\forall n \in \mathbb{N}. P(n) \Rightarrow P(\mathsf{suc}(n))$):

- Given $n \in \mathbb{N}$, let us assume $odd(n) = odd'(n)$ and $even(n) = not(odd'(n))$.
- Second part:

$$
\begin{aligned}
&even(\mathsf{suc}(n)) &&= \\
&odd(n) &&= \{\text{By the first IH.}\} \\
&odd'(n) &&= \\
&not(not(odd'(n))) &&= \\
&not(odd'(\mathsf{suc}(n)))
\end{aligned}
$$

Discuss how you would prove $\forall n \in \mathbb{N}.\ even(n) = nots(n, \mathsf{true})$.

$$
\begin{aligned}
&nots \in \mathbb{N} \times Bool \to Bool \\
&nots(\mathsf{zero}, \quad b) = b \\
&nots(\mathsf{suc}(n), b) = nots(n, not(b))
\end{aligned}
$$

$$odd, even \in \mathbb{N} \to Bool$$

$$
\begin{aligned}
odd(\mathsf{zero}) &= \mathsf{false} \\
odd(\mathsf{suc}(n)) &= even(n)
\end{aligned}
$$

$$
\begin{aligned}
even(\mathsf{zero}) &= \mathsf{true} \\
even(\mathsf{suc}(n)) &= odd(n)
\end{aligned}
$$

Respond at `https://pingo.coactum.de/729558`.

Today

- Proofs.
- Proofs by induction.
- Inductively defined sets.
- Recursive functions.

Next lecture

- Structural induction.
- Some concepts from automata theory.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson,
partly based on slides by Ana Bove

2024-01-22

Today

- Structural induction.
- Some concepts from automata theory.
- Inductively defined subsets
  (if we have time).

Structural induction

Structural induction

- For a given inductively defined set we have a corresponding induction principle.
- Example:

$$\frac{}{\mathsf{zero} \in \mathbb{N}} \qquad\qquad \frac{n \in \mathbb{N}}{\mathsf{suc}(n) \in \mathbb{N}}$$

In order to prove $\forall n \in \mathbb{N}.\ P(n)$:

- Prove $P(\mathsf{zero})$.
- For all $n \in \mathbb{N}$, prove that $P(n)$ implies $P(\mathsf{suc}(n))$.

Structural induction

- For a given inductively defined set we have a corresponding induction principle.
- Example:

$$\frac{}{\mathsf{true} \in Bool} \qquad \frac{}{\mathsf{false} \in Bool}$$

In order to prove $\forall b \in Bool.\ P(b)$:

- Prove $P(\mathsf{true})$.
- Prove $P(\mathsf{false})$.

Structural induction

- For a given inductively defined set we have a corresponding induction principle.
- Example:

$$\frac{}{\mathsf{nil} \in List(A)} \qquad \frac{x \in A \qquad xs \in List(A)}{\mathsf{cons}(x, xs) \in List(A)}$$

In order to prove $\forall xs \in List(A).\ P(xs)$:

- Prove $P(\mathsf{nil})$.
- For all $x \in A$ and $xs \in List(A)$, prove that $P(xs)$ implies $P(\mathsf{cons}(x, xs))$.

Pattern

- An inductively defined set:

$$\ldots \qquad \frac{x \in A \qquad \ldots \qquad d \in D(A)}{\mathsf{c}(x, \ldots, d) \in D(A)} \qquad \ldots$$

  Note that x is a non-recursive argument, and that d is recursive.
- In order to prove $\forall d \in D(A).\ P(d)$:
  - $\vdots$
  - For all $x \in A$, ..., $d \in D(A)$, prove that ... and $P(d)$ imply $P(\mathsf{c}(x, \ldots, d))$.
  - $\vdots$

  One inductive hypothesis for each *recursive* argument.

What is the induction principle for

$$\frac{n \in \mathbb{N}}{\mathsf{leaf}(n) \in \mathit{Tree}} \qquad \frac{l, r \in \mathit{Tree}}{\mathsf{node}(l, r) \in \mathit{Tree}}?$$

1. $(\forall n \in \mathbb{N}.\ P(\mathsf{leaf}(n))) \wedge$
   $(\forall l, r \in \mathit{Tree}.\ P(l) \wedge P(r) \Rightarrow P(\mathsf{node}(l, r)))$.
2. $(\forall n \in \mathbb{N}.\ P(\mathsf{leaf}(n))) \wedge$
   $(\forall l, r \in \mathit{Tree}.\ P(l) \wedge P(r) \Rightarrow P(\mathsf{node}(l, r))) \Rightarrow$
   $(\forall t \in \mathit{Tree}.\ P(t))$.
3. $(\forall n \in \mathbb{N}.\ P(\mathsf{leaf}(n))) \wedge$
   $(\forall t \in \mathit{Tree}.\ P(t) \Rightarrow P(\mathsf{node}(t, t))) \Rightarrow$
   $(\forall t \in \mathit{Tree}.\ P(t))$.

Respond at `https://pingo.coactum.de/729558`.

Some functions

Recall from last lecture:

$$\begin{aligned}
&\mathit{length} \in \mathit{List}(A) \to \mathbb{N} \\
&\mathit{length}(\mathsf{nil}) &&= \mathsf{zero} \\
&\mathit{length}(\mathsf{cons}(x, xs)) &&= \mathsf{suc}(\mathit{length}(xs))
\end{aligned}$$

$$\begin{aligned}
&\mathit{append} \in \mathit{List}(A) \times \mathit{List}(A) \to \mathit{List}(A) \\
&\mathit{append}(\mathsf{nil}, \quad ys) &&= ys \\
&\mathit{append}(\mathsf{cons}(x, xs), ys) &&= \mathsf{cons}(x, \mathit{append}(xs, ys))
\end{aligned}$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Let us prove the property

$$
\begin{aligned}
P(xs) := \forall ys \in List(A).\\
&length(append(xs, ys)) =\\
&length(xs) + length(ys)
\end{aligned}
$$

by induction on the structure of the list.

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case nil:

$$length(append(\mathsf{nil}, ys))$$

$$length(\mathsf{nil}) + length(ys)$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case nil:

$$length(append(\mathsf{nil}, ys)) =$$
$$length(ys)$$

$$length(\mathsf{nil}) + length(ys)$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case nil:

$$
\begin{aligned}
& length(append(\mathsf{nil}, ys)) = \\
& length(ys) \\
& 0 + length(ys) = \\
& length(\mathsf{nil}) + length(ys)
\end{aligned}
$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case nil:

$$
\begin{aligned}
&length(append(\mathsf{nil}, ys)) = \\
&length(ys) = \\
&0 + length(ys) = \\
&length(\mathsf{nil}) + length(ys)
\end{aligned}
$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case $\mathsf{cons}(x, xs)$:

$$length(append(\mathsf{cons}(x, xs), ys))$$

$$length(\mathsf{cons}(x, xs)) + length(ys)$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case $\mathsf{cons}(x, xs)$:

$$length(append(\mathsf{cons}(x, xs), ys)) =$$
$$length(\mathsf{cons}(x, append(xs, ys)))$$

$$length(\mathsf{cons}(x, xs)) + length(ys)$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case $\mathsf{cons}(x, xs)$:

$$length(append(\mathsf{cons}(x, xs), ys)) =$$
$$length(\mathsf{cons}(x, append(xs, ys))) =$$
$$1 + length(append(xs, ys))$$

$$length(\mathsf{cons}(x, xs)) + length(ys)$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case $\mathsf{cons}(x, xs)$:

$$length(append(\mathsf{cons}(x, xs), ys)) =$$
$$length(\mathsf{cons}(x, append(xs, ys))) =$$
$$1 + length(append(xs, ys))$$

$$(1 + length(xs)) + length(ys) =$$
$$length(\mathsf{cons}(x, xs)) + length(ys)$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case $\mathsf{cons}(x, xs)$:

$$
\begin{aligned}
& length(append(\mathsf{cons}(x, xs), ys)) = \\
& length(\mathsf{cons}(x, append(xs, ys))) = \\
& 1 + length(append(xs, ys)) \\
& 1 + (length(xs) + length(ys)) = \\
& (1 + length(xs)) + length(ys) = \\
& length(\mathsf{cons}(x, xs)) + length(ys)
\end{aligned}
$$

Lemma

$\forall xs, ys \in List(A).$
$length(append(xs, ys)) = length(xs) + length(ys).$

Proof.

Case $\mathsf{cons}(x, xs)$:

$$
\begin{aligned}
& length(append(\mathsf{cons}(x, xs), ys)) = \\
& length(\mathsf{cons}(x, append(xs, ys))) = \\
& 1 + length(append(xs, ys)) = \{\text{By the IH, } P(xs).\} \\
& 1 + (length(xs) + length(ys)) = \\
& (1 + length(xs)) + length(ys) = \\
& length(\mathsf{cons}(x, xs)) + length(ys)
\end{aligned}
$$

Prove $\forall xs \in List(A).append(xs, \textsf{nil}) = xs$ and $\forall xs \in List(A).append(\textsf{nil}, xs) = xs$. Which proof is "easiest"?

1. The first.
2. The second.

Respond at `https://pingo.coactum.de/729558`.

Induction/recursion

- Inductively defined sets:
  inference rules with constructors.
- Recursion (primitive recursion):
  recursive calls only for recursive arguments
  ($f(\mathsf{c}(x, d)) = ...f(d)...$).
- Structural induction:
  inductive hypotheses for recursive arguments
  ($P(d) \Rightarrow P(\mathsf{c}(x, d))$).

Some concepts from automata theory

Alphabets and strings

- An *alphabet* is a finite, nonempty set.
  - $\{ a, b, c, ..., z \}$.
  - $\{ 0, 1, ..., 9 \}$.
- A *string (or word) over the alphabet* Σ is a member of $List(\Sigma)$.

Some conventions

Following the course text book:

- Σ: An alphabet.
- a, b, c: Elements of alphabets.
- u, v, w: Words over an alphabet.

Notation

- Σ^* instead of $List(\Sigma)$.
- ε instead of nil or $[\,]$.
- aw instead of cons(a, w).
- a instead of cons(a, nil) or $[a]$.
- abc instead of $[a, b, c]$.
- uv instead of $append(u, v)$.
- $|w|$ instead of $length(w)$.
- Σ^+: Nonempty strings, $\{\, w \in \Sigma^* \mid w \neq \varepsilon \,\}$.

Exponentiation

- Σ^n: Strings of length n, $\{ w \in \Sigma^* \mid |w| = n \}$.
- An example: $\{ a, b \}^2 = \{ aa, ab, ba, bb \}$.
- Alternative definition of $\Sigma^n \subseteq \Sigma^*$:

$$
\begin{aligned}
\Sigma^0 &= \{ \varepsilon \} \\
\Sigma^{n+1} &= \{ aw \mid a \in \Sigma, w \in \Sigma^n \}
\end{aligned}
$$

Exponentiation

- w^n: w repeated n times.
- An example: $(ab)^3 = ababab$.
- A recursive definition:

$$\begin{aligned} w^0 &= \varepsilon \\ w^{n+1} &= ww^n \end{aligned}$$

Which of the following propositions are valid? The alphabet is $\{ a, b, c \}$.

1. $|uv| = |u| + |v|$.
2. $|uv| = |u||v|$.
3. $|w^n| = n$.
4. $uv = vu$.
5. $\varepsilon v = v\varepsilon$.

Respond at `https://pingo.coactum.de/729558`.

Languages

A *language* over an alphabet Σ is a set $L \subseteq \Sigma^*$.

- Typical programming languages.
- Typical natural languages?
  (Are they well-defined?)
- Other examples, for instance the odd natural numbers expressed in binary notation (without leading zeros), which is a language over $\{ 0, 1 \}$.

Another convention

Following the course text book:

- L, M, N: Languages.

Operations

- Concatenation: $LM = \{\, uv \mid u \in L, v \in M \,\}$.
- An example:

$$\{\, a, bc \,\}\{\, de, f \,\} = \{\, ade, af, bcde, bcf \,\}$$

Operations

- Exponentiation:

$$\begin{aligned} L^0 &= \{\, \varepsilon \,\} \\ L^{n+1} &= LL^n \end{aligned}$$

- An example:

$$\begin{aligned} &\{\, a, bc \,\}^2 = \\ &\{\, a, bc \,\}(\{\, a, bc \,\}^1) = \\ &\{\, a, bc \,\}(\{\, a, bc \,\}\{\, \varepsilon \,\}) = \\ &\{\, a, bc \,\}\{\, a, bc \,\} = \\ &\{\, aa, abc, bca, bcbc \,\} \end{aligned}$$

Operations

- Exponentiation:

$$\begin{aligned} L^0 &= \{\ \varepsilon\ \} \\ L^{n+1} &= LL^n \end{aligned}$$

- This definition is consistent with a previous one:

$$\Sigma^n = \{\ w \in \Sigma^* \mid |w| = 1\ \}^n$$

Operations

- The Kleene star $L^* = \bigcup_{n \in \mathbb{N}} L^n$.
- An example:

$$\{\, a, bc \,\}^* =$$
$$\{\, a, bc \,\}^0 \cup \{\, a, bc \,\}^1 \cup \{\, a, bc \,\}^2 \cup \ldots =$$
$$\{\, \varepsilon, a, bc, aa, abc, bca, bcbc, \ldots \,\}$$

- This definition is consistent with a previous one:

$$\Sigma^* = \{\, w \in \Sigma^* \mid |w| = 1 \,\}^*$$

Which of the following propositions are valid? The alphabet is $\{0, 1, 2\}$.

1. $\forall w \in L^n.\ |w| = n$.
2. $LM = ML$.
3. $L(M \cup N) = LM \cup LN$.
4. $LM \cap LN \subseteq L(M \cap N)$.
5. $L^* L^* \subseteq L^*$.

Respond at `https://pingo.coactum.de/729558`.

Which of the following propositions are valid? The alphabet is $\{ 0, 1, 2 \}$.

1. $\forall w \in L^n.\ |w| = n$.

No. Counterexample: $L = \{ \varepsilon \}$, $n = 1$.

Which of the following propositions are valid? The alphabet is $\{ 0, 1, 2 \}$.

2. $LM = ML$.

No. Counterexample: $L = \{ 0 \}$, $M = \{ 1 \}$.

Which of the following propositions are valid? The alphabet is $\{ 0, 1, 2 \}$.

3. $L(M \cup N) = LM \cup LN$.

Yes. The set $L(M \cup N)$ consists exactly of the strings in LM and the strings in LN.

Which of the following propositions are valid? The alphabet is $\{ 0, 1, 2 \}$.

4. $LM \cap LN \subseteq L(M \cap N)$.

No. With $L = \{ \varepsilon, 1 \}$, $M = \{ 1 \}$ and $N = \{ \varepsilon \}$ we get that

$$\begin{aligned}
& LM \cap LN && = \\
& \{ 1, 11 \} \cap \{ \varepsilon, 1 \} && = \\
& \{ 1 \} && \not\subseteq \\
& \emptyset && = \\
& L\emptyset && = \\
& L(M \cap N).
\end{aligned}$$

Which of the following propositions are valid? The alphabet is $\{0, 1, 2\}$.

5. $L^*L^* \subseteq L^*$.

Yes. Any string in L^*L^* consists of

- a string in L^* followed by a string in L^*,
- i.e. m strings in L followed by n strings in L (for some $m, n \in \mathbb{N}$),
- i.e. $m + n$ strings in L,
- and such a string is a member of L^*.

In fact, $(L^*)^* = L^*$.

Inductively defined subsets

Inductively defined subsets

- One can define subsets of (say) Σ^* inductively.
- For instance, for $L \subseteq \Sigma^*$ we can define $L^* \subseteq \Sigma^*$ inductively:

$$\frac{}{\varepsilon \in L^*} \qquad \frac{u \in L \qquad v \in L^*}{uv \in L^*}$$

- Note that there are no constructors (but in some cases it might make sense to name the rules).

$$\frac{}{\varepsilon \in L^*} \qquad \frac{u \in L \qquad v \in L^*}{uv \in L^*}$$

$aba \in \{\, a, ab \,\}^*$

Proof:

$$\dfrac{\dfrac{}{ab \in \{\, a, ab \,\}} \qquad \dfrac{\dfrac{}{a \in \{\, a, ab \,\}} \qquad \dfrac{}{\varepsilon \in \{\, a, ab \,\}^*}}{a \in \{\, a, ab \,\}^*}}{aba \in \{\, a, ab \,\}^*}$$

$bab \notin \{ a, ab \}^*$

Proof:

- Because $bab \neq \varepsilon$ a derivation of $bab \in \{ a, ab \}^*$ would have to end in the following way, with $uv = bab$:

$$\frac{u \in \{ a, ab \} \qquad v \in \{ a, ab \}^*}{uv \in \{ a, ab \}^*}$$

- Because $u \in \{ a, ab \}$ we get that $u = a$ or $u = ab$.
- In either case we get a contradiction, because u must be empty or start with b.

Inductively defined subsets

- What about recursion?

$$
\begin{aligned}
&f \in L^* \to Bool \\
&f(\varepsilon) \quad = \mathsf{false} \\
&f(uv) = not(f(v))
\end{aligned}
$$

- If $\varepsilon \in L$, do we have

$$f(\varepsilon) = f(\varepsilon\varepsilon) = not(f(\varepsilon))?$$

Inductively defined subsets

- Induction works
  (assuming “proof irrelevance”).
- $P(\varepsilon) \wedge (\forall u \in L, v \in L^*.\ P(v) \Rightarrow P(uv)) \Rightarrow$
  $\forall w \in L^*.\ P(w)$.

Another example

$L \subseteq \{ a, b \}^*$ is defined inductively in the following way:

$$\frac{}{a \in L} \qquad\qquad \frac{u, v \in L}{ubv \in L}$$

An induction principle for L:

$$P(a) \wedge (\forall u, v \in L.\ P(u) \wedge P(v) \Rightarrow P(ubv)) \Rightarrow \forall w \in L.\ P(w)$$

$L \subseteq \{ a, b \}^*$ is defined inductively in the following way:

$$\frac{}{a \in L} \qquad \frac{u, v \in L}{ubv \in L}$$

Which of the following propositions are valid?

1. $\varepsilon \in L$.
2. $aba \in L$.
3. $bab \in L$.
4. $aabaa \in L$.
5. $ababa \in L$.

Respond at `https://pingo.coactum.de/729558`.

Today

- Structural induction.
- Some concepts from automata theory.
- Inductively defined subsets.

Next lecture

- Deterministic finite automata.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson,
partly based on slides by Ana Bove

2024-01-23

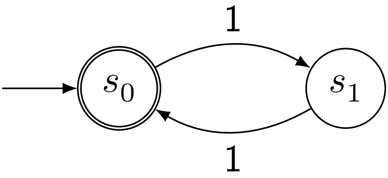
Today

- Deterministic finite automata.

DFAs

DFAs

Recall from the first lecture:



- A DFA specifies a language.
- In this case the language $\{ 11 \}^* = \{ \varepsilon, 11, 1111, \ldots \}$.
- DFAs are for instance used to implement regular expression matching.

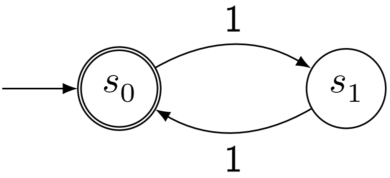
DFAs

A DFA can be given by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$:

- A finite set of states (Q).
- An alphabet (Σ).
- A transition function $(\delta \in Q \times \Sigma \to Q)$.
- A start state $(q_0 \in Q)$.
- A set of accepting states $(F \subseteq Q)$.

DFAs

The diagram



corresponds to the 5-tuple

$$Even = (\{\, s_0, s_1 \,\}, \{\, 1 \,\}, \delta, s_0, \{\, s_0 \,\}),$$

where δ is defined in the following way:

$$\begin{aligned}
&\delta \in \{\, s_0, s_1 \,\} \times \{\, 1 \,\} \to \{\, s_0, s_1 \,\} \\
&\delta(s_0, 1) = s_1 \\
&\delta(s_1, 1) = s_0
\end{aligned}$$

Which of the following 5-tuples can be seen as DFAs?

1. $(\mathbb{N}, \{ 0, 1 \}, \delta, 0, \{ 13 \})$, where $\delta(n, m) = n + m$.
2. $(\{ 0, 1 \}, \emptyset, \delta, 0, \{ 1 \})$, where $\delta(n, \_) = n$.
3. $(\{ q_0, q_1 \}, \{ 0, 1 \}, \delta, q_0, \{ 1 \})$, where $\delta(\_, \_) = q_0$.
4. $(\{ q_0, q_1 \}, \{ 0, 1 \}, \delta, q_0, \{ q_0 \})$, where $\delta(q, \_) = q$.
5. $(\{ q_0, q_1 \}, \{ 0, 1 \}, \delta, q_0, \{ q_0 \})$, where $\delta(\_, \_) = 0$.

Respond at `https://pingo.coactum.de/729558`.

Semantics

The language of a DFA

The language $L(A)$ of a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is defined in the following way:

- A transition function for strings is defined by recursion:

$$\begin{aligned}
&\hat{\delta} \in Q \times \Sigma^* \to Q \\
&\hat{\delta}(q,\, \varepsilon) \quad = q \\
&\hat{\delta}(q,\, aw) = \hat{\delta}(\delta(q, a), w)
\end{aligned}$$

- The language is $\left\{\, w \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, w) \in F \,\right\}$.

The language of a DFA

For $Even$:

$$\begin{aligned}
\hat{\delta}(s_0, 11) &= \\
\hat{\delta}(\delta(s_0, 1), 1) &= \\
\hat{\delta}(s_1, 1) &= \\
\hat{\delta}(\delta(s_1, 1), \varepsilon) &= \\
\hat{\delta}(s_0, \varepsilon) &= \\
s_0 &
\end{aligned}$$

Which strings are members of the language of $(\{ s_0, s_1, s_2, s_3 \}, \{ a, b \}, \delta, s_0, \{ s_0 \})$? Here δ is defined in the following way:

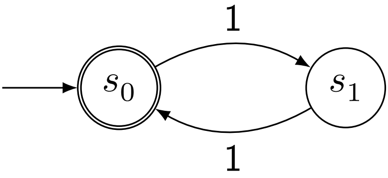
$$\begin{array}{ll} \delta(s_0, a) = s_1 & \delta(s_0, b) = s_2 \\ \delta(s_1, a) = s_0 & \delta(s_2, b) = s_0 \\ \delta(\_, \_) = s_3 & \text{(In all other cases.)} \end{array}$$

1. ε.
2. aab.
3. aba.
4. $aabbaa$.
5. $abbaab$.
6. $bbaaaa$.

Respond at `https://pingo.coactum.de/729558`.

Transition diagrams

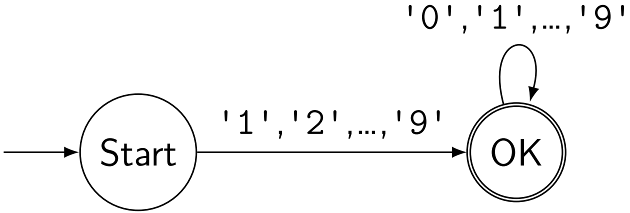
Transition diagrams



- One node per state.
- An arrow "from nowhere" to the start state.
- Double circles for accepting states.
- For every transition $\delta(s_1, a) = s_2$,
  an arrow marked with a from s_1 to s_2.
    - Multiple arrows can be combined.

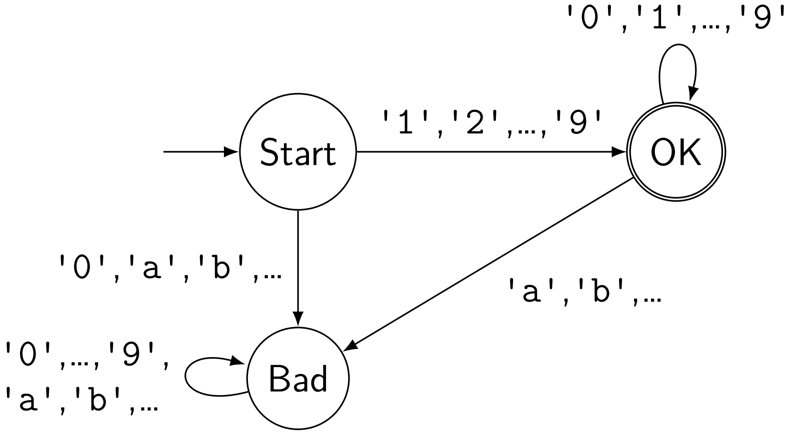
A variant

Diagrams with “missing transitions”:



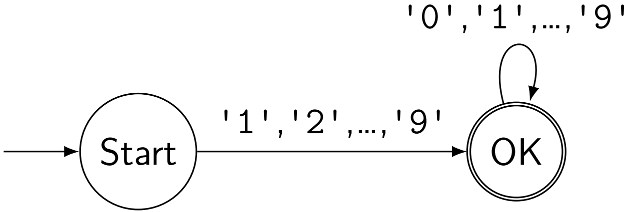
A variant

Every missing transition goes to a new state (that is not accepting):



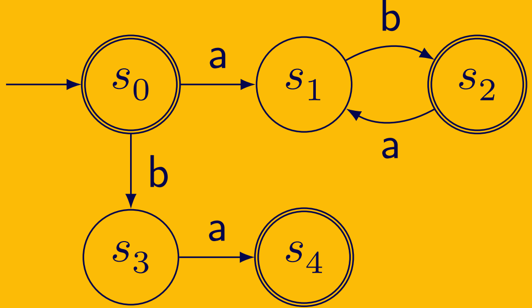
A variant

Note that diagrams with missing transitions do not define the alphabet unambiguously:



The alphabet must be a (finite) superset of { '0', '1', ..., '9' }, but which one?

Which strings are members of the language of the DFA defined by the following transition diagram? The alphabet is $\{ a, b \}$.



1. ε.
2. aa.
3. ab.
4. ba.
5. $abab$.
6. $baba$.

Respond at `https://pingo.coactum.de/729558`.

Transition tables

Transition tables

	0	1
$\rightarrow *s_0$	s_2	s_1
s_1	s_2	s_0
s_2	s_2	s_2

- States: Left column.
- Alphabet: Upper row.
- Start state: Arrow.
- Accepting states: Stars.
- Transition function: Table.

Which strings are members of the language of the DFA defined by the following transition table?

	0	1
$\rightarrow s_0$	s_2	s_1
$*s_1$	s_2	s_0
$*s_2$	s_2	s_2

1. ε.
2. 0.
3. 1.
4. 11.
5. 111.
6. 1010.

Respond at `https://pingo.coactum.de/729558`.

Constructions

Complement

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ we can construct a DFA $\overline{A}$ that satisfies the following property:

$$L(\overline{A}) = \overline{L(A)} := \Sigma^* \setminus L(A).$$

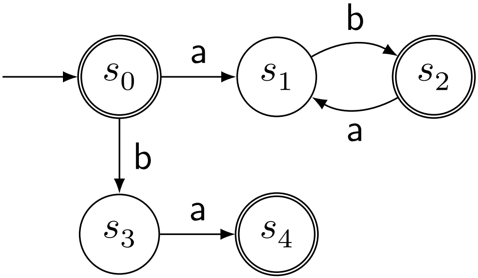
Construction:

$$(Q, \Sigma, \delta, q_0, Q \setminus F).$$

We accept if the original automaton doesn't.

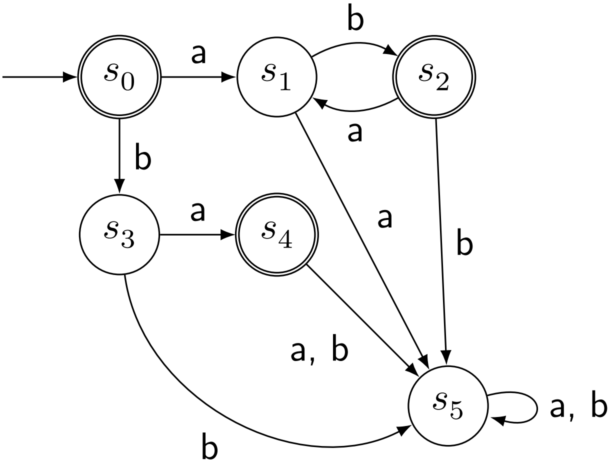
Complement

$A =$



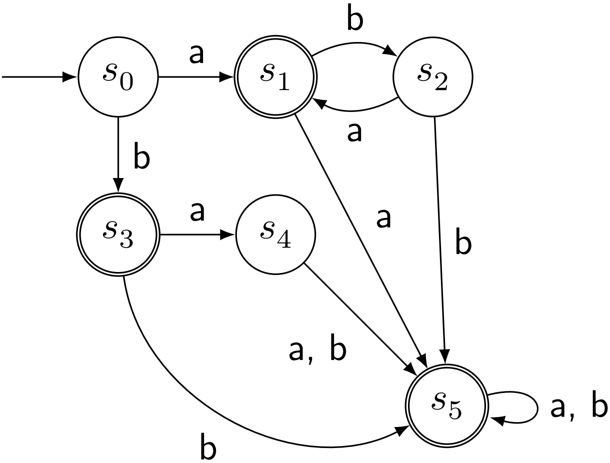
Complement

$A =$



Complement

$\overline{A} =$



Product

Given two DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ with the same alphabet we can construct a DFA $A_1 \otimes A_2$ that satisfies the following property:

$$L(A_1 \otimes A_2) = L(A_1) \cap L(A_2).$$

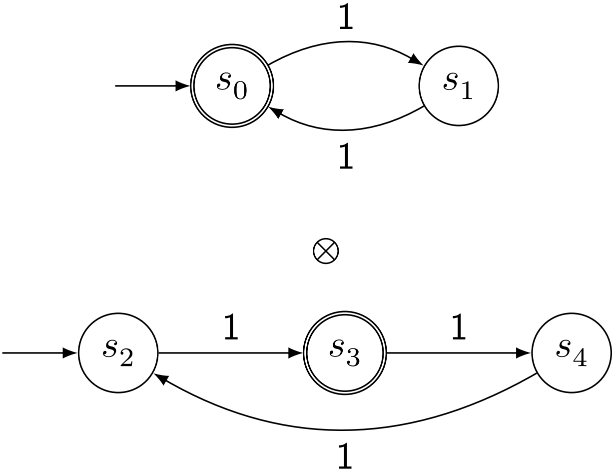
Construction:

$(Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F_1 \times F_2)$, where
$\delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a))$.

We basically run the two automatons in parallel and accept if both accept.

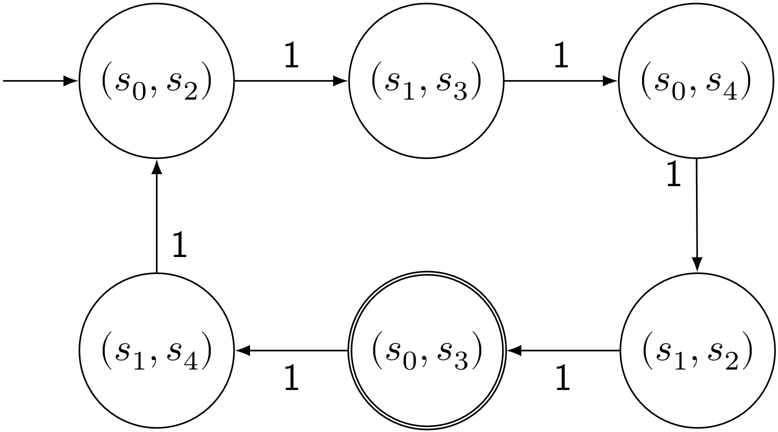
Product

$\{ 2n \mid n \in \mathbb{N} \} \cap \{ 1 + 3n \mid n \in \mathbb{N} \}$
(in unary notation, with ε standing for 0):



Product

$\{ 4 + 6n \mid n \in \mathbb{N} \}$:



We can also construct a DFA $A_1 \oplus A_2$ that satisfies the following property:

$$L(A_1 \oplus A_2) = L(A_1) \cup L(A_2).$$

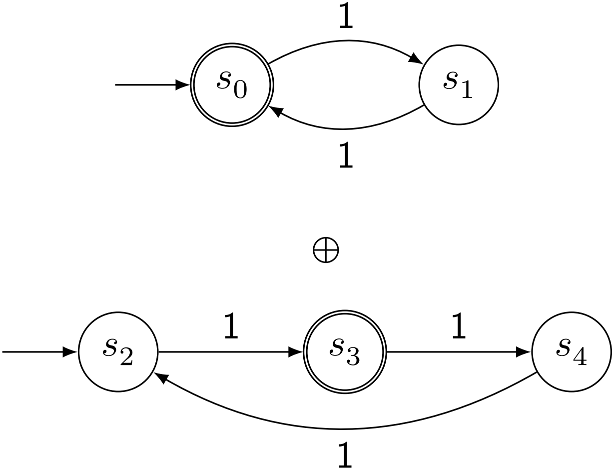
The construction is basically that of $A_1 \otimes A_2$, but with a different set of accepting states. Which one?

1. $F_1 \cup F_2$.
2. $F_1 \cap F_2$.
3. $Q_1 \times Q_2$.
4. $F_1 \times Q_2 \ \cup \ Q_1 \times F_2$.
5. $F_1 \times Q_2 \ \cap \ Q_1 \times F_2$.

Respond at `https://pingo.coactum.de/729558`.

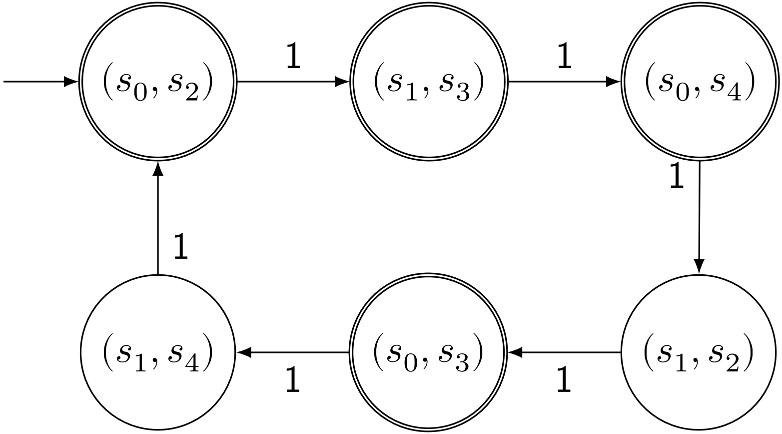
Sum

$\{\, 2n \mid n \in \mathbb{N} \,\} \cup \{\, 1 + 3n \mid n \in \mathbb{N} \,\}$:



Sum

$\{ 2n \mid n \in \mathbb{N} \} \cup \{ 1 + 6n \mid n \in \mathbb{N} \}$:



Accessible states

- Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
- The set $Acc(q) \subseteq Q$ of states that are accessible from $q \in Q$ can be defined in the following way:

$$Acc(q) = \left\{ \hat{\delta}(q, w) \;\middle|\; w \in \Sigma^* \right\}$$

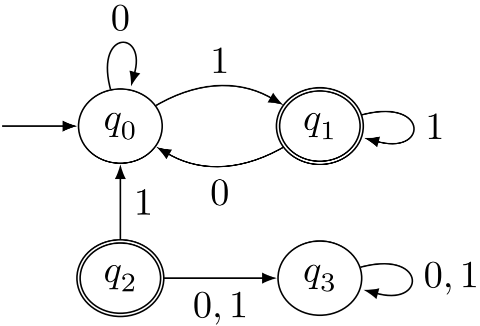
- A possibly smaller DFA:

$$A' = (Acc(q_0), \Sigma, \delta', q_0, F \cap Acc(q_0))$$
$$\delta'(q, a) = \delta(q, a)$$

- We have $L(A') = L(A)$.

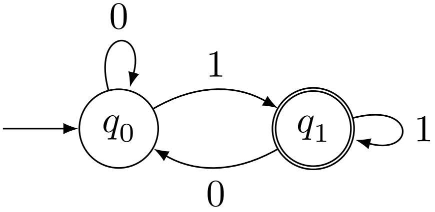
Accessible states

Note that some states cannot be reached from the start state:



Accessible states

The following DFA defines the same language:



Regular languages

Regular languages

- A language $M \subseteq \Sigma^*$ is *regular* if there is some DFA A with alphabet Σ such that $L(A) = M$.
- Note that if M and N are regular, then $M \cap N$, $M \cup N$ and $\overline{M}$ are also regular.
- We will see later that if M and N are regular, then MN is regular.

Which of the following languages are regular? ($\Sigma = \{ 0, 1 \}$.)

1. $\{ w \in \Sigma^* \mid |w| \leq 7 \}$.
2. $\{ w \in \Sigma^* \mid |w| > 7 \}$.
3. $\Sigma^* \{ 11 \} \Sigma^*$.
4. $\{ w \in \Sigma^* \mid \exists u, v \in \Sigma^*.\, w = u11v \}$.
5. $\{ w \in \Sigma^* \mid |w| \leq 7 \vee \exists u, v \in \Sigma^*.\, w = u11v \}$.
6. $\{ w \in \Sigma^* \mid |w| > 7 \wedge \nexists u, v \in \Sigma^*.\, w = u11v \}$.

Today

Deterministic finite automata:

- 5-tuples.
- Semantics.
- Transition diagrams.
- Transition tables.
- Constructions.
- Regular languages.

Next lecture

- Nondeterministic finite automata (NFAs).
- The subset construction
  (turns NFAs into DFAs).

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson,
partly based on slides by Ana Bove

2024-01-25

Today

- Nondeterministic finite automata (NFAs).
- Equivalence of NFAs and DFAs.
- Perhaps something about how one can model things using finite automata.

The first assignment

In the first assignment you are given an inductively defined subset of $\{ a, b \}^*$:

$$\frac{}{\varepsilon \in S} \qquad \frac{u, v \in S}{auavb \in S} \qquad \frac{u, v, w \in S}{buavaw \in S}$$

For this set we get the following induction principle (assuming "proof irrelevance"):

$$\begin{aligned}
&P(\varepsilon) \wedge \\
&(\forall u, v \in S. P(u) \wedge P(v) \Rightarrow P(auavb)) \wedge \\
&(\forall u, v, w \in S. P(u) \wedge P(v) \wedge P(w) \Rightarrow P(buavaw)) \\
&\quad \Rightarrow \\
&\forall w \in S. P(w)
\end{aligned}$$

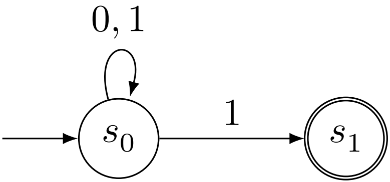
NFAs

NFAs

- Like DFAs, but multiple transitions may be possible.
- An NFA can be in multiple states at once.
- Can be easier to “program”.
- Can be much more compact.

NFAs

Strings over $\{ 0, 1 \}$ that end with a one:



When a one is read the NFA "guesses" whether it should stay in s_0 or go to s_1.

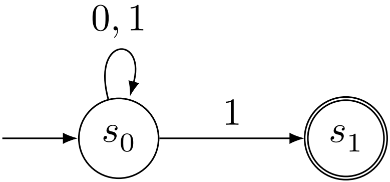
NFAs

An NFA can be given by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$:

- A finite set of states (Q).
- An alphabet (Σ).
- A transition function ($\delta \in Q \times \Sigma \to \wp(Q)$).
- A start state ($q_0 \in Q$).
- A set of accepting states ($F \subseteq Q$).

NFAs

If the alphabet is $\{ 0, 1 \}$, then the diagram



corresponds to the 5-tuple

$$Ends\text{-}with\text{-}one = (\{ s_0, s_1 \}, \{ 0, 1 \}, \delta, s_0, \{ s_1 \}),$$

where δ is defined in the following way:

$$\begin{array}{ll}
\delta \in \{ s_0, s_1 \} \times \{ 0, 1 \} \to \wp(\{ s_0, s_1 \}) & \\
\delta(s_0, 0) = \{ s_0 \} & \delta(s_1, _) = \emptyset \\
\delta(s_0, 1) = \{ s_0, s_1 \} &
\end{array}$$

The language of an NFA

The language $L(A)$ of an NFA $A = (Q, \Sigma, \gamma, q_0, F)$ is defined in the following way:

- A transition function for strings is defined by recursion:

$$
\begin{aligned}
&\hat{\gamma} \in Q \times \Sigma^* \to \wp(Q) \\
&\hat{\gamma}(q, \varepsilon) \quad = \{\, q \,\} \\
&\hat{\gamma}(q, aw) = \textstyle\bigcup_{r \in \gamma(q,a)} \hat{\gamma}(r, w)
\end{aligned}
$$

- The language is

$$
\{\, w \in \Sigma^* \mid \hat{\gamma}(q_0, w) \cap F \neq \emptyset \,\}\,.
$$

The language of an NFA

For *Ends-with-one*:

$$\hat{\delta}(s_0, 10)$$

The language of an NFA

For $\mathit{Ends\text{-}with\text{-}one}$:

$$\hat{\delta}(s_0, 10) \qquad =$$

$$\bigcup_{q \in \delta(s_0,1)} \hat{\delta}(q, 0)$$

The language of an NFA

For $\mathit{Ends\text{-}with\text{-}one}$:

$$\hat{\delta}(s_0, 10) \quad =$$

$$\bigcup_{q \in \{ s_0, s_1 \}} \hat{\delta}(q, 0)$$

The language of an NFA

For *Ends-with-one*:

$$\begin{aligned}
\hat{\delta}(s_0, 10) &= \\
\bigcup_{q \in \{ s_0, s_1 \}} \hat{\delta}(q, 0) &= \\
\bigcup_{q \in \{ s_0, s_1 \}} \bigcup_{r \in \delta(q,0)} \hat{\delta}(r, \varepsilon) &
\end{aligned}$$

The language of an NFA

For *Ends-with-one*:

$$
\begin{aligned}
\hat{\delta}(s_0, 10) &= \\
\bigcup_{q \in \{ s_0, s_1 \}} \hat{\delta}(q, 0) &= \\
\bigcup_{q \in \{ s_0, s_1 \}} \bigcup_{r \in \delta(q,0)} \hat{\delta}(r, \varepsilon) &= \\
\bigcup_{r \in \bigcup_{q \in \{ s_0, s_1 \}} \delta(q,0)} \hat{\delta}(r, \varepsilon)
\end{aligned}
$$

The language of an NFA

For *Ends-with-one*:

$$\begin{aligned}
\hat{\delta}(s_0, 10) &= \\
\bigcup_{q \in \{\, s_0, s_1 \,\}} \hat{\delta}(q, 0) &= \\
\bigcup_{q \in \{\, s_0, s_1 \,\}} \bigcup_{r \in \delta(q,0)} \hat{\delta}(r, \varepsilon) &= \\
\bigcup_{r \in \bigcup_{q \in \{\, s_0, s_1 \,\}} \delta(q,0)} \hat{\delta}(r, \varepsilon) &= \\
\bigcup_{r \in \delta(s_0,0) \cup \delta(s_1,0)} \hat{\delta}(r, \varepsilon)
\end{aligned}$$

The language of an NFA

For *Ends-with-one*:

$$
\begin{aligned}
&\hat{\delta}(s_0, 10) &=\\
&\bigcup_{q \in \{ s_0, s_1 \}} \hat{\delta}(q, 0) &=\\
&\bigcup_{q \in \{ s_0, s_1 \}} \bigcup_{r \in \delta(q,0)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \bigcup_{q \in \{ s_0, s_1 \}} \delta(q,0)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \{ s_0 \} \cup \emptyset} \hat{\delta}(r, \varepsilon)
\end{aligned}
$$

The language of an NFA

For *Ends-with-one*:

$$
\begin{aligned}
&\hat{\delta}(s_0, 10) &=\\
&\bigcup_{q \in \{\, s_0, s_1 \,\}} \hat{\delta}(q, 0) &=\\
&\bigcup_{q \in \{\, s_0, s_1 \,\}} \bigcup_{r \in \delta(q,0)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \bigcup_{q \in \{\, s_0, s_1 \,\}} \delta(q,0)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \{\, s_0 \,\}} \hat{\delta}(r, \varepsilon)
\end{aligned}
$$

The language of an NFA

For *Ends-with-one*:

$$
\begin{aligned}
&\hat{\delta}(s_0, 10) &=\\
&\bigcup_{q \in \{\, s_0, s_1 \,\}} \hat{\delta}(q, 0) &=\\
&\bigcup_{q \in \{\, s_0, s_1 \,\}} \bigcup_{r \in \delta(q,0)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \bigcup_{q \in \{\, s_0, s_1 \,\}} \delta(q,0)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \{\, s_0 \,\}} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \{\, s_0 \,\}} \{\, r \,\}
\end{aligned}
$$

The language of an NFA

For *Ends-with-one*:

$$
\begin{aligned}
&\hat{\delta}(s_0, 10) &=\\
&\bigcup_{q \in \{ s_0, s_1 \}} \hat{\delta}(q, 0) &=\\
&\bigcup_{q \in \{ s_0, s_1 \}} \bigcup_{r \in \delta(q,0)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \bigcup_{q \in \{ s_0, s_1 \}} \delta(q,0)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \{ s_0 \}} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \{ s_0 \}} \{ r \} &=\\
&\{ s_0 \}
\end{aligned}
$$

Transition diagrams

As for DFAs, but with one change:

- For every transition $\delta(s_1, a) = S$
  and every state $s_2 \in S$,
  an arrow marked with a from s_1 to s_2.

Note:

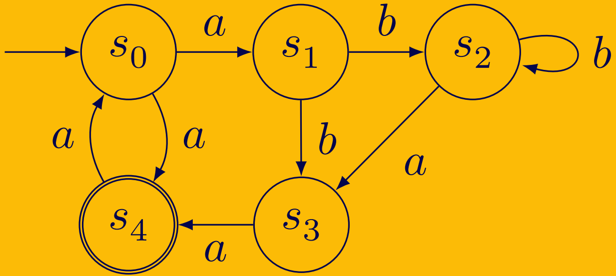
- The alphabet is not defined unambiguously.
- No need for special treatment of missing transitions, because $\delta(s_1, a)$ can be empty.

Transition tables

As for DFAs, but with one change:

- The result of a transition is a set of states instead of a state.

Which strings are members of the language of the following NFA over $\{ a, b, c \}$?



1. $abba$.
2. $abbaca$.
3. $aaabaa$.
4. $aaabaaa$.
5. $aaaabaa$.
6. $abbaaabaaa$.

Respond at `https://pingo.coactum.de/729558`.

Some conventions

At least partly following the course text book:

- q, r, s: A state.
- δ: A transition function.

Which of the following propositions are valid?

1. $\hat{\delta}(q, a) = \delta(q, a)$.
2. $\hat{\delta}(q, uv) = \hat{\delta}(q, vu)$.
3. $\hat{\delta}(q, uv) = \bigcup_{r \in \hat{\delta}(q,v)} \hat{\delta}(r, u)$.
4. $\hat{\delta}(q, uv) = \bigcup_{r \in \hat{\delta}(q,u)} \hat{\delta}(r, v)$.

You may want to use the following lemma:

$$\bigcup_{y \in \bigcup_{x \in X} F(x)} G(y) = \bigcup_{x \in X} \bigcup_{y \in F(x)} G(y)$$

Respond at `https://pingo.coactum.de/729558`.

Which of the following propositions are valid?

1. $\hat{\delta}(q, a) = \delta(q, a)$.

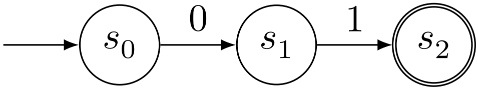
Yes:

$$
\begin{aligned}
&\hat{\delta}(q, a) &=\\
&\bigcup_{r \in \delta(q,a)} \hat{\delta}(r, \varepsilon) &=\\
&\bigcup_{r \in \delta(q,a)} \{\, r \,\} &=\\
&\delta(q, a)
\end{aligned}
$$

Which of the following propositions are valid?

2. $\hat{\delta}(q, uv) = \hat{\delta}(q, vu)$.

No. Counterexample:



Denote the transition function by δ.

$$\hat{\delta}(s_0, 01) = \{ s_2 \} \neq \emptyset = \hat{\delta}(s_0, 10)$$

Which of the following propositions are valid?

4. $\hat{\delta}(q, uv) = \bigcup_{r \in \hat{\delta}(q,u)} \hat{\delta}(r, v)$.

Yes. Proof by induction on the structure of the string u:

$$\begin{aligned}
&\hat{\delta}(q, \varepsilon v) &&= \\
&\hat{\delta}(q, v) &&= \\
&\bigcup_{r \in \{\, q \,\}} \hat{\delta}(r, v) &&= \\
&\bigcup_{r \in \hat{\delta}(q,\varepsilon)} \hat{\delta}(r, v)
\end{aligned}$$

Which of the following propositions are valid?

4. $\hat{\delta}(q, uv) = \bigcup_{r \in \hat{\delta}(q,u)} \hat{\delta}(r, v)$.

Yes. Proof by induction on the structure of the string u:

$$\hat{\delta}(q, auv) =$$

$$\bigcup_{r' \in \delta(q,a)} \hat{\delta}(r', uv) =$$

$$\bigcup_{r' \in \delta(q,a)} \bigcup_{r \in \hat{\delta}(r',u)} \hat{\delta}(r, v) =$$

$$\bigcup_{r \in \bigcup_{r' \in \delta(q,a)} \hat{\delta}(r',u)} \hat{\delta}(r, v) =$$

$$\bigcup_{r \in \hat{\delta}(q,au)} \hat{\delta}(r, v)$$

Which of the following propositions are valid?

3. $\hat{\delta}(q, uv) = \bigcup_{r \in \hat{\delta}(q,v)} \hat{\delta}(r, u)$.

No, we have

$$\bigcup_{r \in \hat{\delta}(q,v)} \hat{\delta}(r, u) = \hat{\delta}(q, vu),$$

which in general is not equal to $\hat{\delta}(q, uv)$.

NFAs versus DFAs

NFAs versus DFAs

- Every DFA can be seen as an NFA:
  - Turn $\delta(s_1, a) = s_2$ into $\delta(s_1, a) = \{ s_2 \}$.
- Thus every language that can be defined by a DFA can also be defined by an NFA.
- What about the other direction?
  Are NFAs more powerful?
- No.

Subset construction

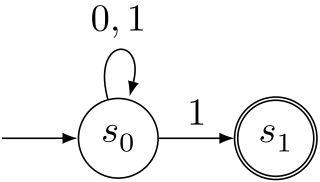
Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$ we can define a DFA D with the same alphabet in such a way that $L(N) = L(D)$:

$$D = (\wp(Q), \Sigma, \delta', \{\, q_0 \,\}, \{\, S \subseteq Q \mid S \cap F \neq \emptyset \,\})$$
$$\delta'(S, a) = \bigcup_{s \in S} \delta(s, a)$$

- The DFA keeps track of exactly which states the NFA is in.
- It accepts if at least one of the NFA states is accepting.

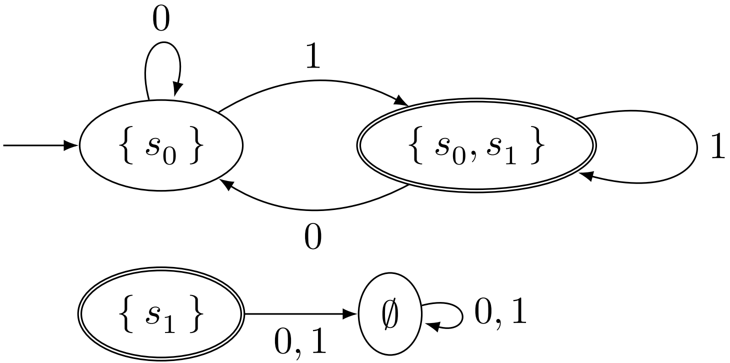
Subset construction

An NFA:



Subset construction

If we apply the subset construction we get the following DFA:

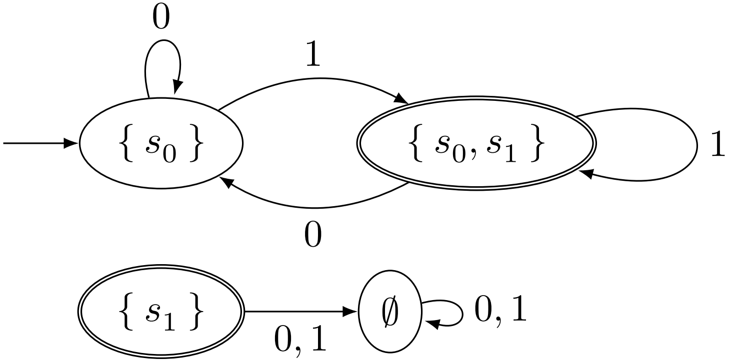


If an NFA has 10 states, and we use the subset construction to build a corresponding DFA, how many states does the DFA have?

Respond at `https://pingo.coactum.de/729558`.

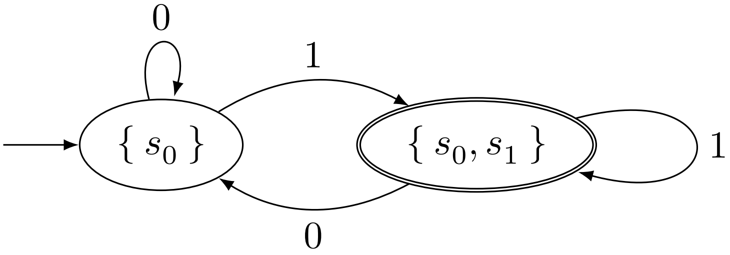
Accessible states

Note that some states cannot be reached from the start state:



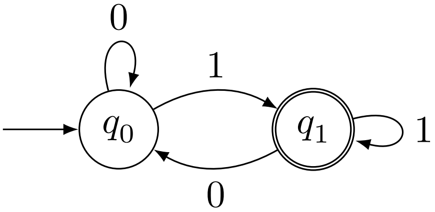
Accessible states

If we remove non-accessible states, then we get a DFA which defines the same language:



Accessible states

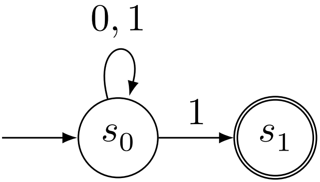
One can also rename the states:



Subset construction

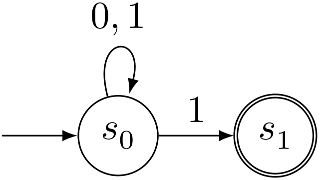
- Note that one does not have to first construct a DFA with $2^{|Q|}$ states, and then remove inaccessible states.
- One can instead construct the DFA without inaccessible states right away:
  - Start with the start state.
  - Add new states reachable from the start state.
  - Add new states reachable from those states.
  - And so on until there are no more new states.

Subset construction



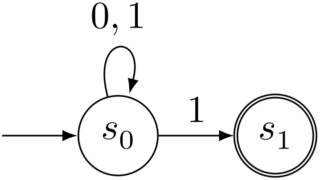
	0	1
$\rightarrow \{\ s_0\ \}$		

Subset construction



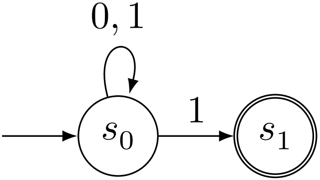
	0	1
$\rightarrow \{\, s_0 \,\}$	$\{\, s_0 \,\}$	

Subset construction



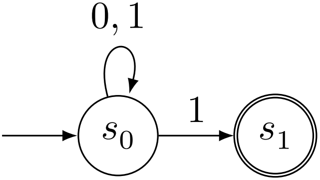
	0	1
$\rightarrow \{ s_0 \}$	$\{ s_0 \}$	$\{ s_0, s_1 \}$
$* \{ s_0, s_1 \}$		

Subset construction



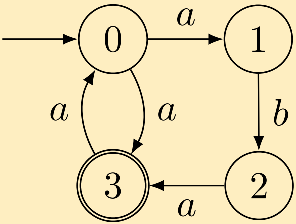
	0	1
$\rightarrow \{ s_0 \}$	$\{ s_0 \}$	$\{ s_0, s_1 \}$
$* \{ s_0, s_1 \}$	$\{ s_0 \}$	

Subset construction



	0	1
$\rightarrow \{ s_0 \}$	$\{ s_0 \}$	$\{ s_0, s_1 \}$
$* \{ s_0, s_1 \}$	$\{ s_0 \}$	$\{ s_0, s_1 \}$

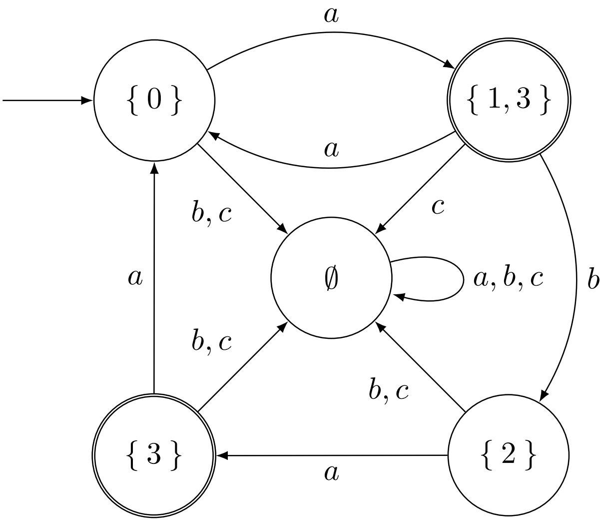
If the subset construction is used to build a DFA corresponding to the following NFA over $\{ a, b, c \}$, and inaccessible states are removed, how many states are there in the resulting DFA?



Respond at `https://pingo.coactum.de/729558`.

How many states are there in the resulting DFA?

5:



Subset construction

Recall the subset construction for $N = (Q, \Sigma, \delta, q_0, F)$:

$$D = (\wp(Q), \Sigma, \delta', \{ q_0 \}, \{ S \subseteq Q \mid S \cap F \neq \emptyset \})$$
$$\delta'(S, a) = \bigcup_{s \in S} \delta(s, a)$$

How would you prove $L(N) = L(D)$?

$$L(N) = \left\{ w \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, w) \cap F \neq \emptyset \right\}$$
$$L(D) = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} \widehat{\delta'}(\{ q_0 \}, w) \in \\ \quad \{ S \subseteq Q \mid S \cap F \neq \emptyset \} \end{array} \right\}$$

Subset construction

Recall the subset construction for $N = (Q, \Sigma, \delta, q_0, F)$:

$$D = (\wp(Q), \Sigma, \delta', \{ q_0 \}, \{ S \subseteq Q \mid S \cap F \neq \emptyset \})$$
$$\delta'(S, a) = \bigcup_{s \in S} \delta(s, a)$$

How would you prove $L(N) = L(D)$?

$$L(N) = \left\{ w \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, w) \cap F \neq \emptyset \right\}$$
$$L(D) = \left\{ w \in \Sigma^* \;\middle|\; \widehat{\delta'}(\{ q_0 \}, w) \cap F \neq \emptyset \right\}$$

Subset construction

This follows from

$$\forall w \in \Sigma^*.\ \forall q \in Q.\ \hat{\delta}(q, w) = \widehat{\delta'}(\{\, q \,\}, w),$$

which can be proved by induction on the structure of the string, using the following lemma:

$$\forall w \in \Sigma^*.\ \forall S \subseteq Q.\ \widehat{\delta'}(S, w) = \bigcup_{s \in S} \widehat{\delta'}(\{\, s \,\}, w)$$

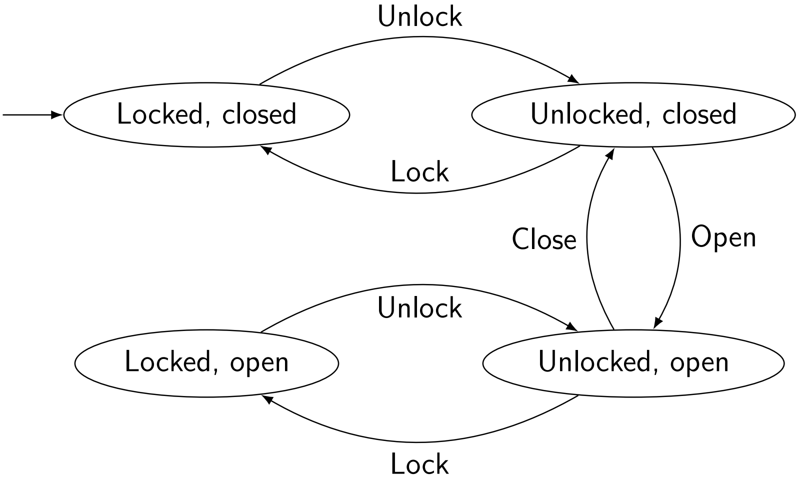
The lemma can also be proved by induction on the structure of the string.

Regular languages

- Recall that a language $M \subseteq \Sigma^*$ is regular if there is some DFA A with alphabet Σ such that $L(A) = M$.
- A language $M \subseteq \Sigma^*$ is also regular if there is some *NFA* A with alphabet Σ such that $L(A) = M$.

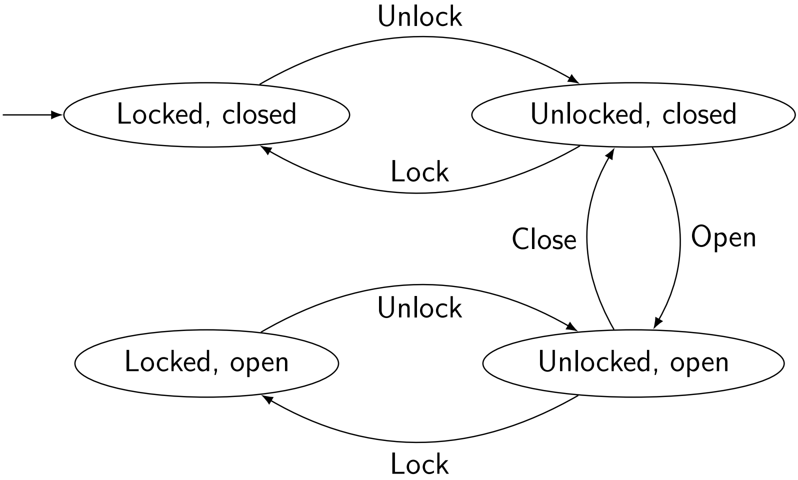
Models

A model of a door



Alphabet: { Lock, Unlock, Open, Close }.

A model of a door



What happens if we try to lock a locked door? Does the system "crash"?

Try to model something as a finite automaton:

- The traffic lights of an intersection.
- A coin-operated vending machine.
- ...

How well does your model work? Does it make sense to model the phenomenon as a finite automaton?

Today

- Nondeterministic finite automata (NFAs).
- The subset construction.
- Models.

Consultation time

- Tomorrow.
- You decide what you want to work on.

Next lecture

Nondeterministic finite automata with ε-transitions.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-01-29

Today

- NFAs with ε-transitions.
- Exponential blowup.

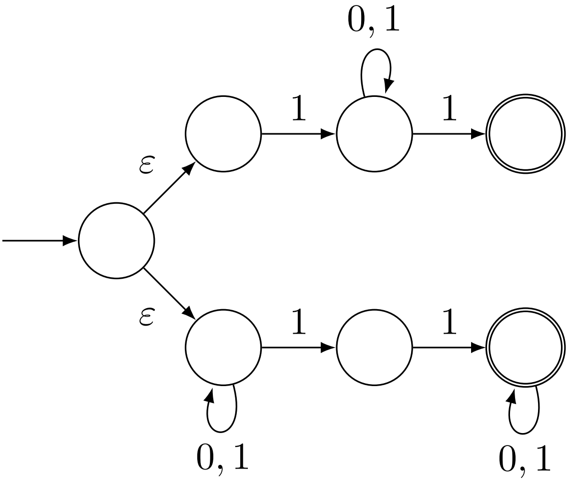
ε-NFAs

ε-NFAs

- Like NFAs, but with ε-transitions:
  The automaton can "spontaneously" make a transition from one state to another.
- Can be used to convert regular expressions to finite automata.

ε-NFAs

Strings over $\{ 0, 1 \}$ that start and end with a one, or that contain two consecutive ones:



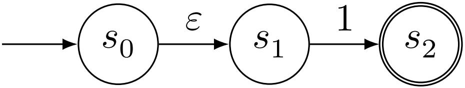
ε-NFAs

An ε-NFA can be given by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$:

- A finite set of states (Q).
- An alphabet (Σ with $\varepsilon \notin \Sigma$).
- A transition function ($\delta \in Q \times (\Sigma \cup \{ \varepsilon \}) \to \wp(Q)$).
- A start state ($q_0 \in Q$).
- A set of accepting states ($F \subseteq Q$).

ε-NFAs

If the alphabet is $\{\ 1\ \}$, then the diagram



corresponds to the 5-tuple

$$One = (\{\ s_0, s_1, s_2\ \}, \{\ 1\ \}, \delta, s_0, \{\ s_2\ \}),$$

where δ is defined in the following way:

$$\begin{array}{lll}
\delta \in \{\ s_0, s_1, s_2\ \} \times \{\ \varepsilon, 1\ \} \to \wp(\{\ s_0, s_1, s_2\ \}) & & \\
\delta(s_0, \varepsilon) = \{\ s_1\ \} & \delta(s_1, \varepsilon) = \emptyset & \delta(s_2, _) = \emptyset \\
\delta(s_0, 1) = \emptyset & \delta(s_1, 1) = \{\ s_2\ \} &
\end{array}$$

Transition diagrams

As for NFAs, but arrows can be labelled with ε.

Transition tables

As for NFAs, but with one column for ε.

ε-closure

ε-closure

The ε-closure of a state q consists of those states that one can reach from q by following zero or more ε-transitions.

ε-closure

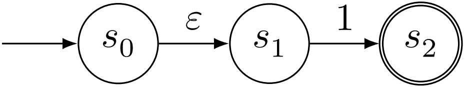
Given an ε-NFA $A = (Q, \Sigma, \delta, q_0, F)$ one can, for each state $q \in Q$, define the ε-closure of q (a subset of Q) inductively in the following way:

$$\frac{}{q \in \varepsilon\text{-}closure(q)}$$

$$\frac{q' \in \varepsilon\text{-}closure(q) \qquad q'' \in \delta(q', \varepsilon)}{q'' \in \varepsilon\text{-}closure(q)}$$

ε-closure

Consider the following ε-NFA again:



The set $\varepsilon\text{-}closure(s_0)$ contains two states:

$$\frac{}{s_0 \in \varepsilon\text{-}closure(s_0)}$$

$$\frac{\dfrac{}{s_0 \in \varepsilon\text{-}closure(s_0)} \qquad \dfrac{}{s_1 \in \delta(s_0, \varepsilon)}}{s_1 \in \varepsilon\text{-}closure(s_0)}$$

Some notation

The ε-closure of a set $S \subseteq Q$:

$$\varepsilon\text{-}closure(S) = \bigcup_{s \in S} \varepsilon\text{-}closure(s)$$

Transition functions applied to a set $S \subseteq Q$:

$$\delta(S, a) = \bigcup_{s \in S} \delta(s, a)$$

$$\hat{\delta}(S, w) = \bigcup_{s \in S} \hat{\delta}(s, w)$$

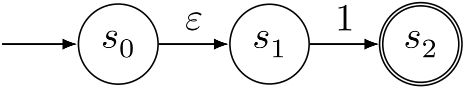
Computing the ε-closure

The ε-closure of q can be computed (perhaps not very efficiently) in the following way:

- Initialise C to $\{ q \}$.
- Repeat until $\delta(C, \varepsilon) \subseteq C$:
  - Set C to $C \cup \delta(C, \varepsilon)$.
- Return C.

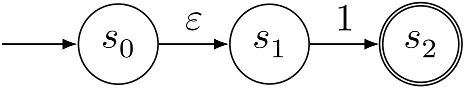
Computing the ε-closure

Let us compute $\varepsilon\text{-}closure(s_0)$ for the following ε-NFA:



Computing the ε-closure

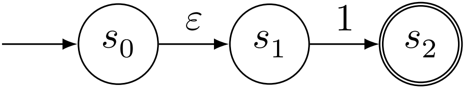
Let us compute $\varepsilon\text{-}closure(s_0)$ for the following ε-NFA:



- Initialise C to $\{ s_0 \}$.

Computing the ε-closure

Let us compute $\varepsilon\text{-}closure(s_0)$ for the following ε-NFA:



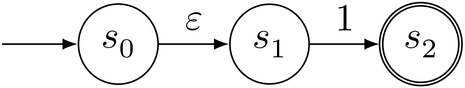
- We have $\delta(C, \varepsilon) \nsubseteq C$:

$$\begin{aligned} \delta(C, \varepsilon) = \delta(\{\, s_0 \,\}, \varepsilon) &= \delta(s_0, \varepsilon) = \\ \{\, s_1 \,\} \;\; \nsubseteq \{\, s_0 \,\} \qquad &= C. \end{aligned}$$

- Set C to $C \cup \delta(C, \varepsilon) = \{\, s_0, s_1 \,\}$.

Computing the ε-closure

Let us compute $\varepsilon\text{-}closure(s_0)$ for the following ε-NFA:

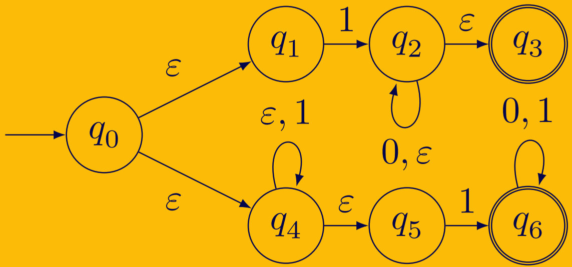


- We have $\delta(C, \varepsilon) \subseteq C$:

$$
\begin{array}{lll}
\delta(C, \varepsilon) & = \delta(\{\, s_0, s_1 \,\}, \varepsilon) & = \\
\delta(s_0, \varepsilon) \cup \delta(s_1, \varepsilon) & = \{\, s_1 \,\} \cup \emptyset & = \\
\{\, s_1 \,\} & \subseteq \{\, s_0, s_1 \,\} & = C.
\end{array}
$$

- Return C.

Which of the following propositions hold for the following ε-NFA over $\{ 0, 1 \}$?



1. $q_0 \in \varepsilon\text{-}closure(q_0)$.
2. $q_5 \in \varepsilon\text{-}closure(q_0)$.
3. $\varepsilon\text{-}closure(q_4) \subseteq \varepsilon\text{-}closure(q_0)$.
4. $q_6 \in \varepsilon\text{-}closure(q_0)$.
5. $q_3 \in \varepsilon\text{-}closure(q_1)$.
6. $\varepsilon\text{-}closure(q_4) \subseteq \varepsilon\text{-}closure(q_5)$.

Respond at `https://pingo.coactum.de/729558`.

Semantics

The language of an ε-NFA

The language $L(A)$ of an ε-NFA $A = (Q, \Sigma, \delta, q_0, F)$ is defined in the following way:

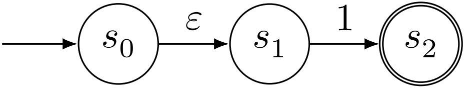
- A transition function for strings is defined by recursion:

$$
\begin{aligned}
&\hat{\delta} \in Q \times \Sigma^* \to \wp(Q) \\
&\hat{\delta}(q, \varepsilon) \quad = \varepsilon\text{-}closure(q) \\
&\hat{\delta}(q, aw) = \hat{\delta}(\delta(\varepsilon\text{-}closure(q), a), w)
\end{aligned}
$$

- The language is

$$
\left\{ w \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, w) \cap F \neq \emptyset \right\}.
$$

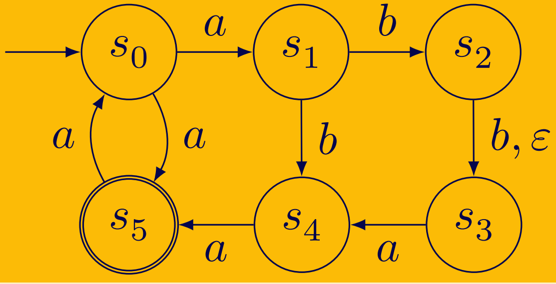
The language of an ε-NFA



For One:

$$
\begin{aligned}
\hat{\delta}(s_0, 1) &= \\
\hat{\delta}(\delta(\varepsilon\text{-}closure(s_0), 1), \varepsilon) &= \\
\hat{\delta}(\delta(\{\ s_0, s_1\ \}, 1), \varepsilon) &= \\
\hat{\delta}(\{\ s_2\ \}, \varepsilon) &= \\
\hat{\delta}(s_2, \varepsilon) &= \\
\varepsilon\text{-}closure(s_2) &= \\
\{\ s_2\ \} &
\end{aligned}
$$

Which strings are members of the language of the following ε-NFA over $\{ a, b, c \}$?



1. $abba$.
2. $abbaca$.
3. $aaabaa$.
4. $aaabaaa$.
5. $aaaabaa$.
6. $abbaaaabaa$.

Respond at `https://pingo.coactum.de/729558`.

Which of the following propositions are valid?

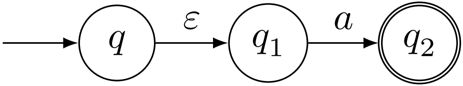
1. $\varepsilon\text{-}closure(\varepsilon\text{-}closure(q)) = \varepsilon\text{-}closure(q)$.
2. $\hat{\delta}(q, w) = \hat{\delta}(\varepsilon\text{-}closure(q), w)$.
3. $\hat{\delta}(\delta(\varepsilon\text{-}closure(q), a), w) =$
   $\hat{\delta}(\varepsilon\text{-}closure(\delta(q, a)), w)$.

Respond at `https://pingo.coactum.de/729558`.

Which of the following propositions are valid?

3. $\hat{\delta}(\delta(\varepsilon\text{-}closure(q), a), w) = \hat{\delta}(\varepsilon\text{-}closure(\delta(q, a)), w)$.

No. Counterexample:



Denote the transition function by δ.

$$\hat{\delta}(\delta(\varepsilon\text{-}closure(q), a), \varepsilon) = \{ q_2 \} \neq \emptyset = \hat{\delta}(\varepsilon\text{-}closure(\delta(q, a)), \varepsilon)$$

Constructions

Subset construction

Given an ε-NFA $N = (Q, \Sigma, \delta, q_0, F)$ we can define a DFA D with the same alphabet in such a way that $L(N) = L(D)$:

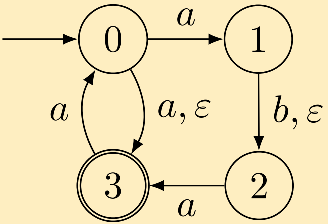
$$D = (\wp(Q), \Sigma, \delta', \varepsilon\text{-}closure(q_0), F')$$

$$\delta'(S, a) = \varepsilon\text{-}closure(\delta(S, a))$$

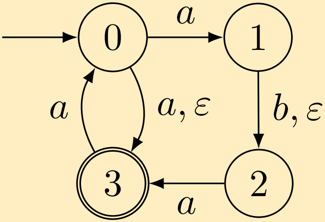
$$F' = \{\, S \subseteq Q \mid S \cap F \neq \emptyset \,\}$$

Every accessible state S is *ε-closed* (i.e. $S = \varepsilon\text{-}closure(S)$).

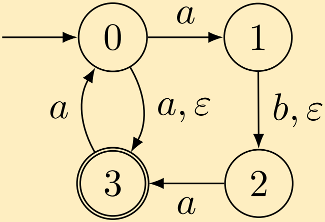
If the subset construction is used to build a DFA corresponding to the following ε-NFA over $\{ a, b \}$, and inaccessible states are removed, how many states are there in the resulting DFA?



Respond at `https://pingo.coactum.de/729558`.



	a	b
$\rightarrow * \{0,3\}$	$\{0,1,2,3\}$	$\emptyset$
$* \{0,1,2,3\}$	$\{0,1,2,3\}$	$\{2\}$
$\emptyset$	$\emptyset$	$\emptyset$
$\{2\}$	$\{3\}$	$\emptyset$
$* \{3\}$	$\{0,3\}$	$\emptyset$



	a	b
$\rightarrow *A$	B	C
$*B$	B	D
C	C	C
D	E	C
$*E$	A	C

Regular languages

- Recall that a language $M \subseteq \Sigma^*$ is regular if there is some DFA (or NFA) A with alphabet Σ such that $L(A) = M$.
- For alphabets Σ with $\varepsilon \notin \Sigma$ a language $M \subseteq \Sigma^*$ is also regular if and only if there is some ε-*NFA* A with alphabet Σ such that $L(A) = M$.

Union

Recall:

- One can use ε-NFAs to convert regular expressions to finite automata.

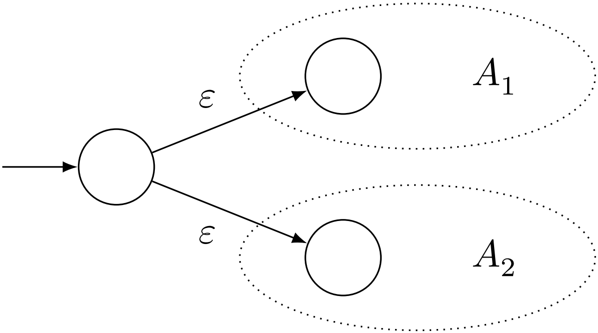
Union

Given two ε-NFAs A_1 and A_2 with the same alphabet we can construct an ε-NFA $A_1 \oplus A_2$ that satisfies the following property:

$$L(A_1 \oplus A_2) = L(A_1) \cup L(A_2).$$

Union

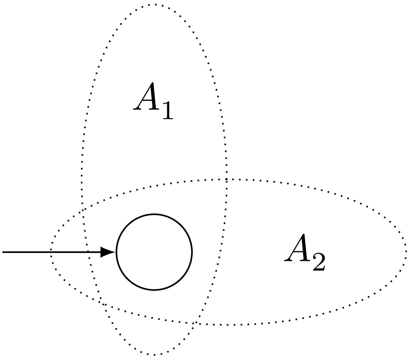
Construction:



- The transitions go to the start states.
- States are renamed if the state sets overlap.

Union

Can one do something similar for NFAs by "merging" the start states?



Given two NFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ satisfying $Q_1 \cap Q_2 = \emptyset$ and $q_0 \notin Q_1 \cup Q_2$, is the language of the NFA

$$(f(Q_1 \cup Q_2), \Sigma, \delta, q_0, f(F_1 \cup F_2)), \text{ where}$$

$$f(S) = (S \setminus \{ q_{01}, q_{02} \}) \cup \{ q_0 \mid q_{01} \in S \vee q_{02} \in S \},$$

$$\delta(s, a) = \begin{cases} f(\delta_1(q_{01}, a) \cup \delta_2(q_{02}, a)), & \text{if } s = q_0, \\ f(\delta_1(s, a)), & \text{if } s \in Q_1, \\ f(\delta_2(s, a)), & \text{if } s \in Q_2 \end{cases}$$

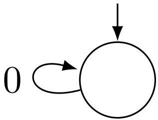
equal to $L(A_1) \cup L(A_2)$?

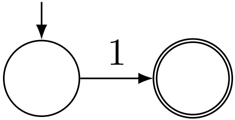
1. Yes, always.
2. No, never.
3. No, not always, but sometimes.

Respond at `https://pingo.coactum.de/729558`.

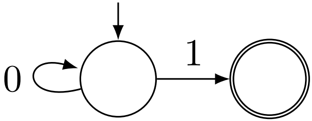
Can one do something similar for NFAs by “merging” the start states?

- Sometimes. For instance if F_1 and F_2 are empty.
- Not always. The following NFAs over $\{ 0, 1 \}$ accept $\emptyset$ and $\{ 1 \}$:





The combination accepts $\{ 0^n 1 \mid n \in \mathbb{N} \}$:



Exponential blowup

Exponential blowup

Consider the following family of languages:

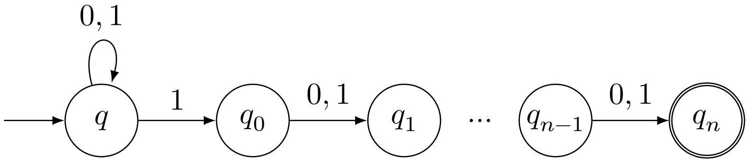
$$A \in \mathbb{N} \to \wp(\{\, 0, 1 \,\}^*)$$
$$A(n) = \left\{\, u1v \;\middle|\; u, v \in \{\, 0, 1 \,\}^*, |v| = n \,\right\}$$

Exponential blowup

The family:

$$A(n) = \{\, u1v \mid u, v \in \{\, 0, 1 \,\}^*, |v| = n \,\}$$

For every $n \in \mathbb{N}$ the NFAs for $A(n)$ with the least number of states have at most $n + 2$ states:



Exponential blowup

Furthermore one can prove:

- For every $n \in \mathbb{N}$ the DFAs for $A(n)$ with the least number of states have at least 2^{n+1} states.

A key part of the proof in the course text book uses the pigeonhole principle:

- A DFA over $\{ 0, 1 \}$ with less than 2^k states has to end up in the same state for at least two distinct k-bit strings.

Exponential blowup

Thus it might be inefficient to check if a string belongs to a language represented by an NFA (or ε-NFA) by using the following method:

- Translate the NFA to a corresponding DFA.
- Use the DFA to check if the string belongs to the language.

Exponential blowup

- This method is used in practice by some tools.
- It seems to work fine in many practical cases.
- Exercise (optional):
  Make such a tool “blow up” by giving it a short piece of carefully crafted input.

Today

- ε-NFAs.
- ε-closure.
- Semantics.
- Constructions.
- Exponential blowup.

Next lecture

- Regular expressions.
- Translation from finite automata to regular expressions.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-02-01

Today

- Regular expressions.
- Translation from finite automata to regular expressions.

Syntax of regular expressions

Syntax

The set $RE(\Sigma)$ of regular expressions over the alphabet Σ can be defined inductively in the following way:

$$\frac{}{\mathsf{empty} \in RE(\Sigma)} \qquad \frac{}{\mathsf{nil} \in RE(\Sigma)}$$

$$\frac{a \in \Sigma}{\mathsf{sym}(a) \in RE(\Sigma)} \qquad \frac{e_1, e_2 \in RE(\Sigma)}{\mathsf{seq}(e_1, e_2) \in RE(\Sigma)}$$

$$\frac{e_1, e_2 \in RE(\Sigma)}{\mathsf{alt}(e_1, e_2) \in RE(\Sigma)} \qquad \frac{e \in RE(\Sigma)}{\mathsf{star}(e) \in RE(\Sigma)}$$

Syntax

Typically we use the following concrete syntax:

$$\frac{}{\emptyset \in RE(\Sigma)} \qquad \frac{}{\varepsilon \in RE(\Sigma)}$$

$$\frac{a \in \Sigma}{a \in RE(\Sigma)} \qquad \frac{e_1, e_2 \in RE(\Sigma)}{e_1 e_2 \in RE(\Sigma)}$$

$$\frac{e_1, e_2 \in RE(\Sigma)}{e_1 + e_2 \in RE(\Sigma)} \qquad \frac{e \in RE(\Sigma)}{e^* \in RE(\Sigma)}$$

(Sometimes $e_1 \mid e_2$ instead of $e_1 + e_2$.)

Syntax

- What if, say, $\varepsilon \in \Sigma$?
- Does ε stand for $\mathsf{sym}(\varepsilon)$ or nil?
- One option: Require that $\emptyset, \varepsilon, +, ^* \notin \Sigma$.

Syntax

- What does $01 + 2$ mean, $(01) + 2$ or $0(1 + 2)$?
- Sequencing "binds tighter" than alternation, so it means $(01) + 2$.
- Parentheses can be used to get the other meaning: $0(1 + 2)$.
- The Kleene star operator binds tighter than sequencing, so 01^* means $0(1^*)$, not $(01)^*$.

Syntax

- What does $0 + 1 + 2$ mean, $0 + (1 + 2)$ or $(0 + 1) + 2$?
- The latter two expressions denote the same language, so the choice is not very important.
- One option (taken by the book): Make the operator left associative, i.e. choose $(0 + 1) + 2$.
- Similarly 012 means $(01)2$.

Syntax

A convention:

- e: A regular expression.

Syntax

An abbreviation:

- e^+ means ee^*.
- This operator binds as tightly as the Kleene star operator.

Which of the following statements are correct?

1. $01 + 23$ means $(01) + (23)$.
2. $01 + 23^*$ means $((01) + (23))^*$.
3. $0 + 1^*2 + 3^*$ means $((0 + 1)^*)((2 + 3)^*)$.
4. $0 + 1^*2 + 3^*$ means $(0 + ((1^*)2)) + (3^*)$.
5. 012^*34 means $((((01)(2^*))3)4)$.

Respond at `https://pingo.coactum.de/729558`.

Semantics

Semantics

$$
\begin{aligned}
&L \in RE(\Sigma) \to \wp(\Sigma^*)\\
&L(\emptyset) &&= \emptyset\\
&L(\varepsilon) &&= \{\, \varepsilon \,\}\\
&L(a) &&= \{\, a \,\}\\
&L(e_1 e_2) &&= L(e_1) L(e_2)\\
&L(e_1 + e_2) &&= L(e_1) \cup L(e_2)\\
&L(e^*) &&= (L(e))^*
\end{aligned}
$$

Semantics

An example:

$$\begin{aligned} L(a + b^*) &= \\ L(a) \cup L(b^*) &= \\ L(a) \cup L(b)^* &= \\ \{\, a \,\} \cup \{\, b \,\}^* & \end{aligned}$$

Which of the following statements are correct?

1. $abcabc \in L(abc^*)$.
2. $xyyxxy \in L(x(y+x)^*y)$.
3. $\varepsilon \in L(\emptyset^*)$.
4. $110 \in L((\emptyset 1 + 10)^*)$.
5. $\varepsilon \in L((\varepsilon + 10)^+)$.
6. $11100 \in L\left(\left(1(0+\varepsilon)\right)^*\right)$.

Respond at `https://pingo.coactum.de/729558`.

Regular expression algebra

Regular expression equivalences

- The equation $e_1 = e_2$ stands for $L(e_1) = L(e_2)$.
- Recall that two languages are equal if they contain the same strings.

Which of the following propositions are valid? The alphabet is $\{ 0, 1 \}$.

1. $e + \emptyset = e$.
2. $e\emptyset = e$.
3. $\varepsilon e = e$.
4. $e_1 e_2 = e_2 e_1$.
5. $e_1 + e_2 = e_2 + e_1$.
6. $e + e = e$.
7. $e_1(e_2 + e_3) = e_1 e_2 + e_1 e_3$.
8. $e_1 + e_2 e_3 = (e_1 + e_2)(e_1 + e_3)$.

Respond at `https://pingo.coactum.de/729558`.

Regular expression algebra

Regular expressions form a semiring:

$$
\begin{aligned}
&e + \emptyset = \emptyset + e = e \\
&e_1 + e_2 = e_2 + e_1 \\
&e_1 + (e_2 + e_3) = (e_1 + e_2) + e_3
\end{aligned}
$$

$$
\begin{aligned}
&e\varepsilon = \varepsilon e = e \\
&e_1(e_2 e_3) = (e_1 e_2) e_3
\end{aligned}
$$

$$
\begin{aligned}
&e\emptyset = \emptyset e = \emptyset \\
&e_1(e_2 + e_3) = e_1 e_2 + e_1 e_3 \\
&(e_1 + e_2) e_3 = e_1 e_3 + e_2 e_3
\end{aligned}
$$

Regular expression algebra

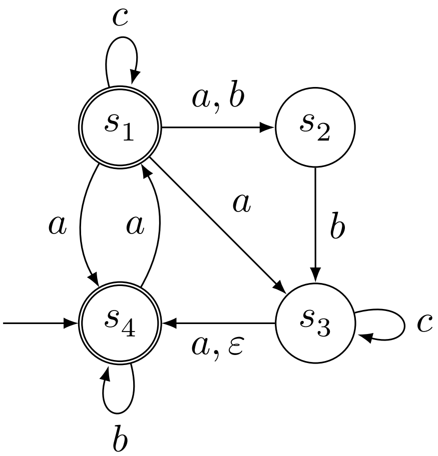
The semiring is idempotent:

$$e + e = e$$

Translating FAs to regular expressions, I

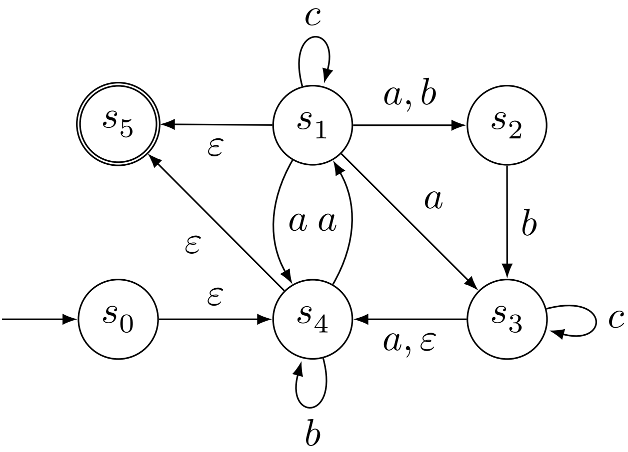
Method one

Consider the following ε-NFA over $\{ a, b, c \}$:



Method one

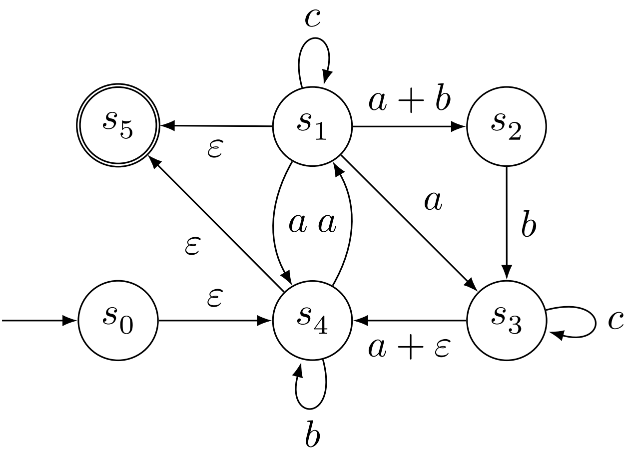
Switch to an equivalent ε-NFA:



(I found this trick in slides due to Klaus Sutner.)

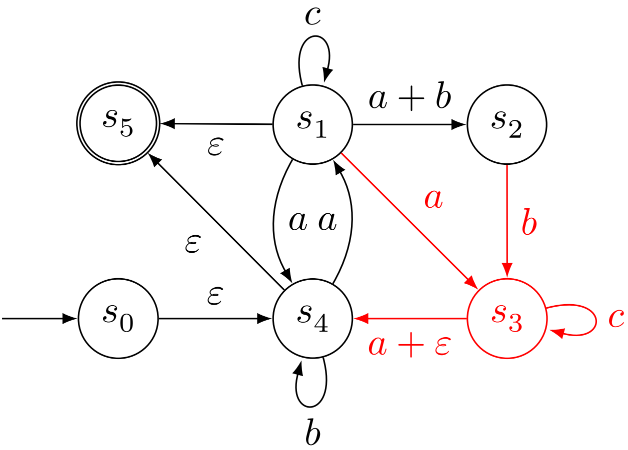
Method one

Turn edge labels into regular expressions:



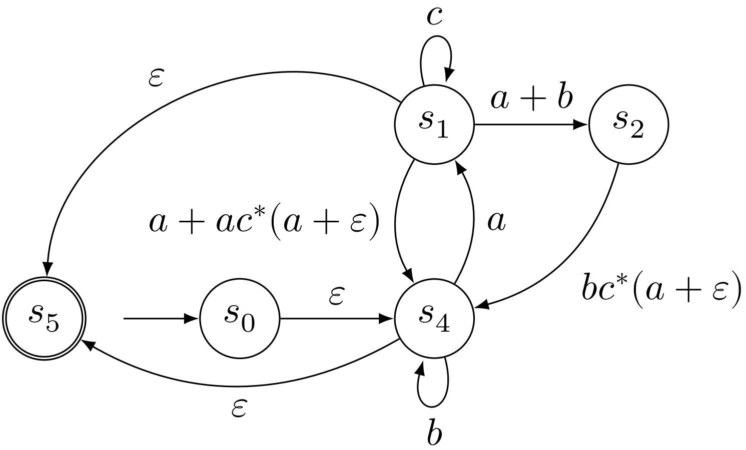
Method one

Eliminate non-accepting states distinct from the start state:



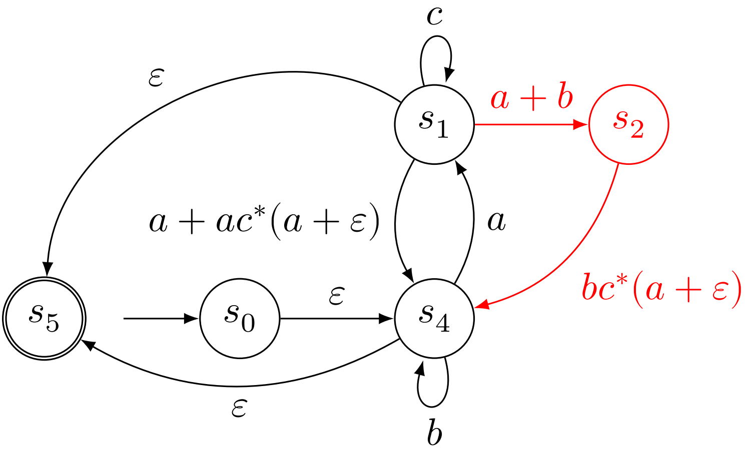
Method one

Eliminate non-accepting states distinct from the start state:



Method one

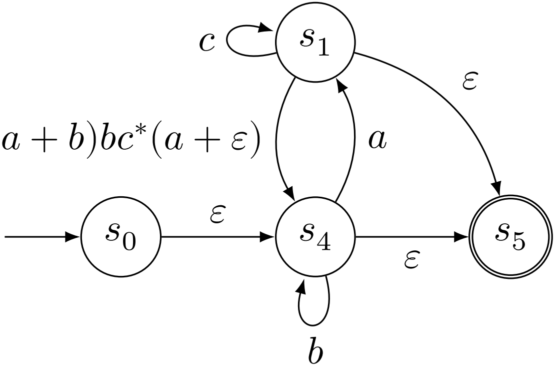
Eliminate non-accepting states distinct from the start state:



Method one

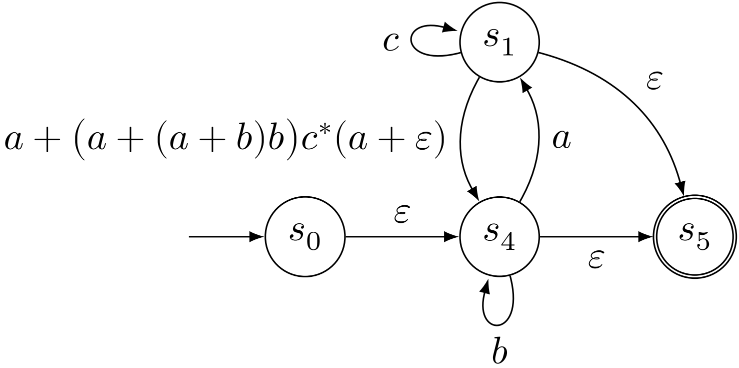
Eliminate non-accepting states distinct from the start state:

$$a + ac^*(a+\varepsilon) + (a+b)bc^*(a+\varepsilon)$$



Method one

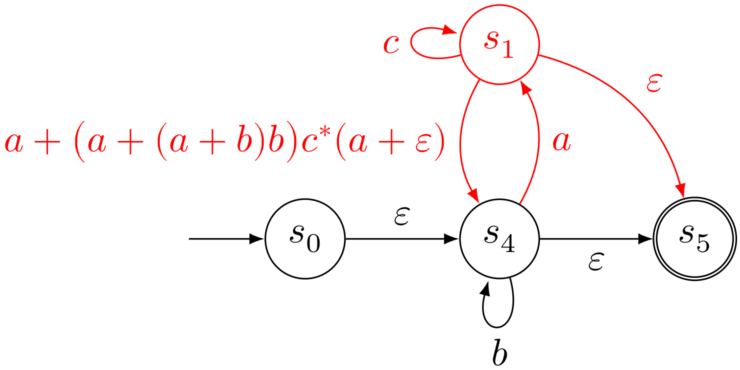
Eliminate non-accepting states distinct from the start state:



It is fine to simplify expressions.

Method one

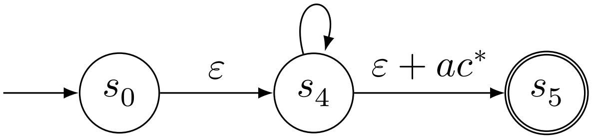
Eliminate non-accepting states distinct from the start state:



Method one

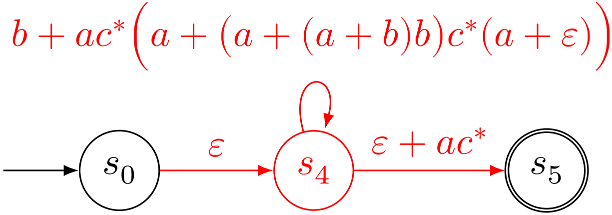
Eliminate non-accepting states distinct from the start state:

$$b + ac^*\Big(a + (a + (a + b)b)c^*(a + \varepsilon)\Big)$$



Method one

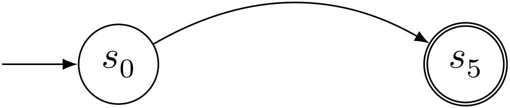
Eliminate non-accepting states distinct from the start state:



Method one

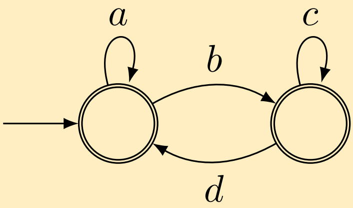
Eliminate non-accepting states distinct from the start state:

$$\left(b + ac^*\left(a + (a + (a + b)b)c^*(a + \varepsilon)\right)\right)^*(\varepsilon + ac^*)$$



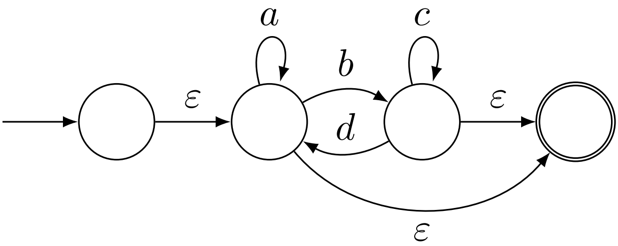
Done.

Turn the following ε-NFA over $\{ a, b, c, d \}$ into a regular expression.

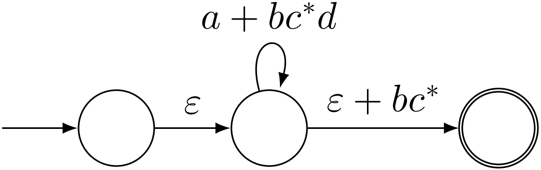


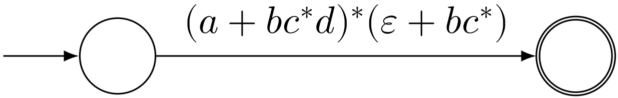
Respond at `https://pingo.coactum.de/729558`.

The result of the first step:



The result of one possible second step:





This is not the only correct solution. Another one:

$$a^*\big(\varepsilon+b(c+da^*b)^*(\varepsilon+da^*)\big)$$

Translating FAs to regular expressions, II

Method two

One form of *Arden's lemma*:

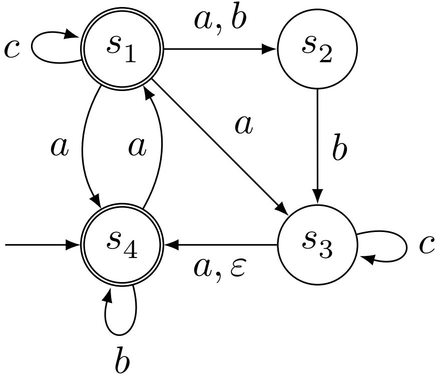
- Let $A, B \subseteq \Sigma^*$ for some alphabet Σ.
- Consider the equation $X = AX \cup B$, where X is restricted to be a subset of Σ^*.
- The equation has the solution $X = A^*B$:

$$A(A^*B) \cup B = (AA^* \cup \{ \varepsilon \})B = A^*B$$

- This solution is the least one (for every other solution Y we have $A^*B \subseteq Y$).
- If $\varepsilon \notin A$, then this solution is unique.

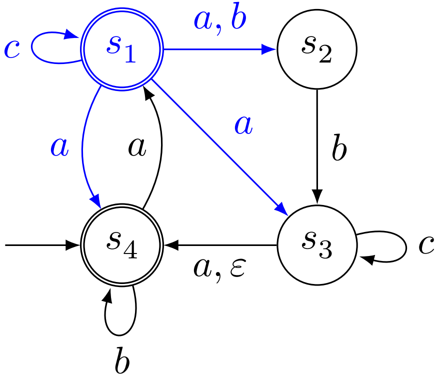
Method two

Consider the following ε-NFA again:



Method two

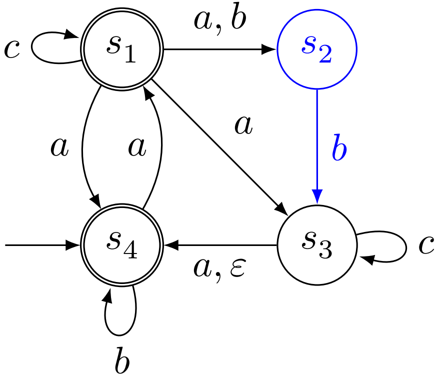
We can turn this ε-NFA into a set of equations.



$$e_1 = \varepsilon + ce_1 + (a+b)e_2 + ae_3 + ae_4$$

Method two

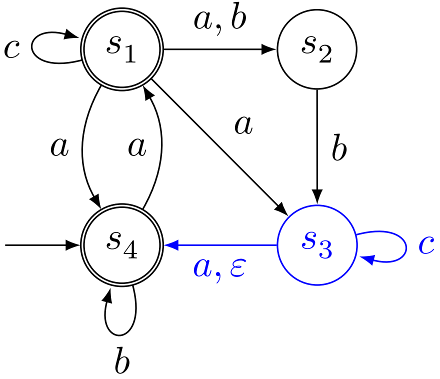
We can turn this ε-NFA into a set of equations.



$$e_2 = be_3$$

Method two

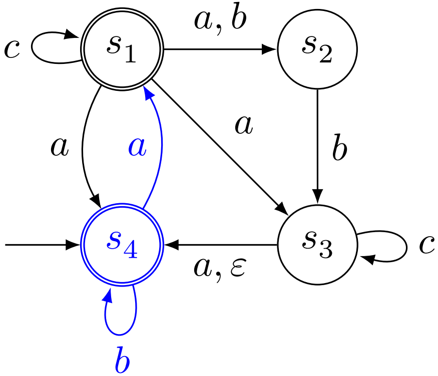
We can turn this ε-NFA into a set of equations.



$$e_3 = ce_3 + (a + \varepsilon)e_4$$

Method two

We can turn this ε-NFA into a set of equations.



$$e_4 = \varepsilon + ae_1 + be_4$$

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$
\begin{aligned}
e_1 &= \varepsilon + ce_1 + (a+b)e_2 + ae_3 + ae_4 \\
e_2 &= be_3 \\
e_3 &= ce_3 + (a+\varepsilon)e_4 \\
e_4 &= \varepsilon + ae_1 + be_4
\end{aligned}
$$

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$
\begin{aligned}
e_1 &= ce_1 + (\varepsilon + (a+b)e_2 + ae_3 + ae_4) \\
e_2 &= be_3 \\
e_3 &= ce_3 + (a+\varepsilon)e_4 \\
e_4 &= be_4 + (\varepsilon + ae_1)
\end{aligned}
$$

Eliminate e_2.

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$
\begin{aligned}
e_1 &= ce_1 + \big(\varepsilon + (a+b)be_3 + ae_3 + ae_4\big)\\
e_3 &= ce_3 + (a+\varepsilon)e_4\\
e_4 &= be_4 + (\varepsilon + ae_1)
\end{aligned}
$$

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$\begin{aligned}
e_1 &= ce_1 + \Big(\varepsilon + (a + (a+b)b)e_3 + ae_4\Big) \\
e_3 &= ce_3 + (a+\varepsilon)e_4 \\
e_4 &= be_4 + (\varepsilon + ae_1)
\end{aligned}$$

Eliminate e_3.

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$\begin{aligned}
e_1 &= ce_1 + \Big(\varepsilon + (a + (a+b)b)e_3 + ae_4\Big) \\
e_3 &= c^*(a+\varepsilon)e_4 \\
e_4 &= be_4 + (\varepsilon + ae_1)
\end{aligned}$$

Eliminate e_3.

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$e_1 = ce_1 + \Big(\varepsilon + (a + (a+b)b)c^*(a+\varepsilon)e_4 + ae_4\Big)$$
$$e_4 = be_4 + (\varepsilon + ae_1)$$

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$e_1 = ce_1 + \left(\varepsilon + \Big(a + (a + (a+b)b)c^*(a+\varepsilon)\Big)e_4\right)$$
$$e_4 = be_4 + (\varepsilon + ae_1)$$

Eliminate e_1.

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$e_1 = c^*\Big(\varepsilon + \big(a + (a + (a+b)b)c^*(a+\varepsilon)\big)e_4\Big)$$
$$e_4 = be_4 + (\varepsilon + ae_1)$$

Eliminate e_1.

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$\begin{aligned} e_4 = {} & be_4 + \varepsilon + \\ & ac^*\Big(\varepsilon + \big(a + (a + (a + b)b)c^*(a + \varepsilon)\big)e_4\Big) \end{aligned}$$

Solve the final equation.

Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$e_4 = \Big(b + ac^*\big(a + (a + (a + b)b)c^*(a + \varepsilon)\big)\Big)e_4 + (\varepsilon + ac^*)$$

Solve the final equation.

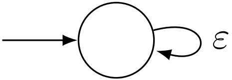
Method two

Goal: Find the *least* solution for e_4.
(Note that e_4 corresponds to the start state.)

$$e_4 = \left(b + ac^*\left(a + (a + (a+b)b)c^*(a+\varepsilon)\right)\right)^*(\varepsilon + ac^*)$$

Method two

- Why the least solution?
- Consider the following ε-NFA:



- The corresponding equation: $e = \varepsilon e$.
- This equation has infinitely many solutions.
- The least solution gives the right answer:

$$e = \varepsilon^* \emptyset = \emptyset$$

Be careful

Consider the following equations:

$$
\begin{aligned}
e_0 &= e_1 \\
e_1 &= ae_1 + be_2 \\
e_2 &= \varepsilon + be_1
\end{aligned}
$$

An incorrect elimination of e_1:

$$
\begin{aligned}
e_0 &= e_1 \\
e_1 &= e_0 \\
e_2 &= \varepsilon + be_1
\end{aligned}
$$

Be careful

Consider the following equations:

$$e_0 = e_1$$
$$e_1 = ae_1 + be_2$$
$$e_2 = \varepsilon + be_1$$

An incorrect elimination of e_1:

$$e_0 = e_0$$
$$e_2 = \varepsilon + be_0$$

Be careful

Consider the following equations:

$$
\begin{aligned}
e_0 &= e_1 \\
e_1 &= ae_1 + be_2 \\
e_2 &= \varepsilon + be_1
\end{aligned}
$$

Use Arden's lemma:

$$e_0 = \varepsilon^* \emptyset = \emptyset$$

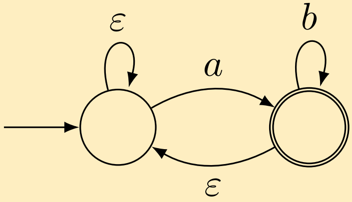
A correct solution:

$$e_0 = a^* b (b a^* b)^*$$

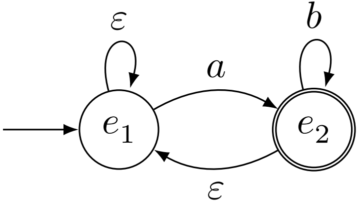
A warning

- A variable stands for the set of strings that take you from the corresponding state to any accepting state.
- Some online videos use a different method, in which a variable corresponding to state s stands for the strings that take you from the start state to state s.

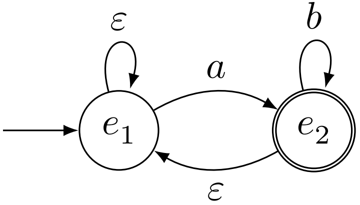
Turn the following ε-NFA over $\{ a, b \}$ into a regular expression.



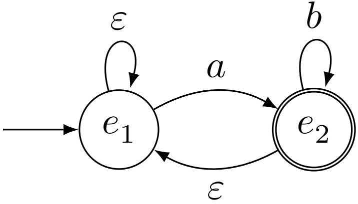
Respond at `https://pingo.coactum.de/729558`.



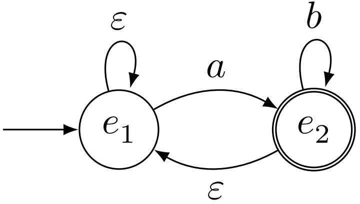
$$
\begin{aligned}
e_1 &= \varepsilon e_1 + a e_2 \\
e_2 &= \varepsilon + b e_2 + \varepsilon e_1
\end{aligned}
$$



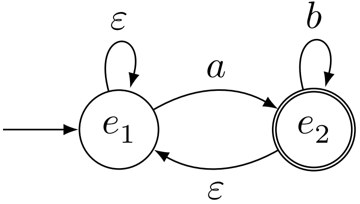
$$e_1 = e_1 + ae_2$$
$$e_2 = be_2 + \varepsilon + e_1$$



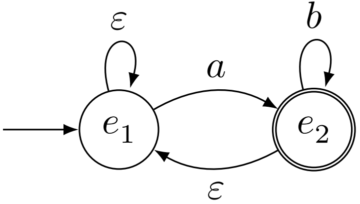
$$e_1 = e_1 + ae_2$$
$$e_2 = b^*(\varepsilon + e_1)$$



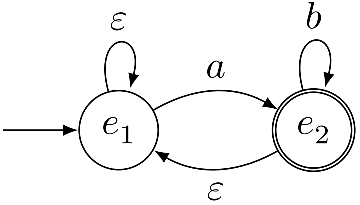
$$e_1 = e_1 + ab^*(\varepsilon + e_1)$$



$$e_1 = (\varepsilon + ab^*)e_1 + ab^*$$

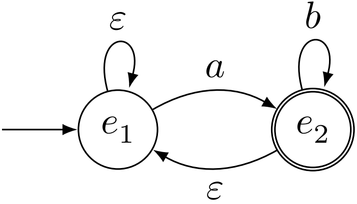


$$e_1 = (\varepsilon + ab^*)^* ab^*$$



Note that $(\varepsilon + e)^* = e^*$:

$$e_1 = (ab^*)^* ab^*$$



Note that $e^*e = e^+$:

$$e_1 = (ab^*)^+$$

Today

- Syntax of regular expressions.
- Semantics of regular expressions.
- Regular expression algebra.
- Two methods for translating finite automata to regular expressions.

Next lecture

- Translation from regular expressions to finite automata.
- More about regular expression algebra.
- The pumping lemma for regular languages.
- Some closure properties for regular languages.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-02-05

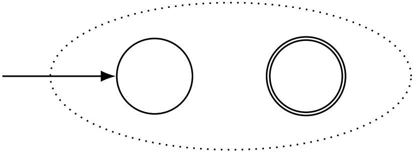
Today

- Translation from regular expressions to finite automata.
- More about regular expression algebra.
- The pumping lemma for regular languages.
- Some closure properties for regular languages.

Translating regular expressions to automata

Regular expressions to automata

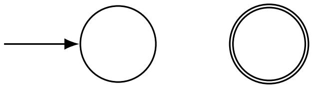
Given a regular expression in $RE(\Sigma)$ we construct an ε-NFA (with alphabet Σ) with exactly one accepting state, no transitions from the accepting state, and no transitions to the start state:



The translation is defined recursively.

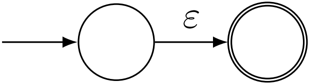
The empty language

$\varepsilon\text{-}NFA(\emptyset) =$



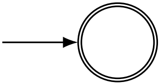
The empty string

$$\varepsilon\text{-}NFA(\varepsilon) =$$



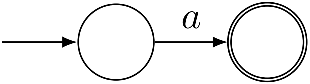
The empty string

$\varepsilon\text{-}NFA'(\varepsilon) =$



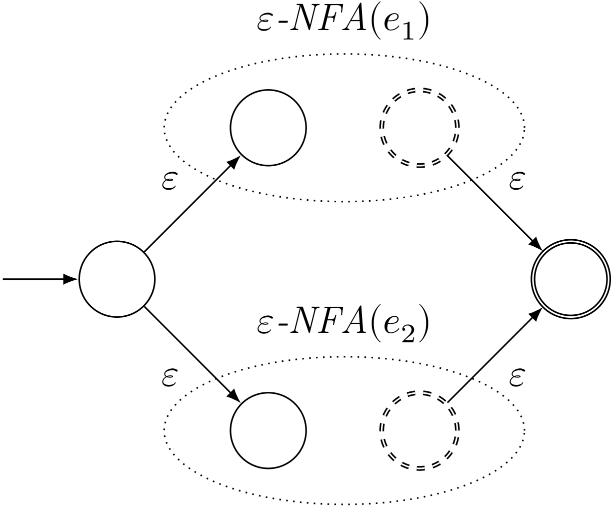
A symbol

$$\varepsilon\text{-}NFA(a) =$$



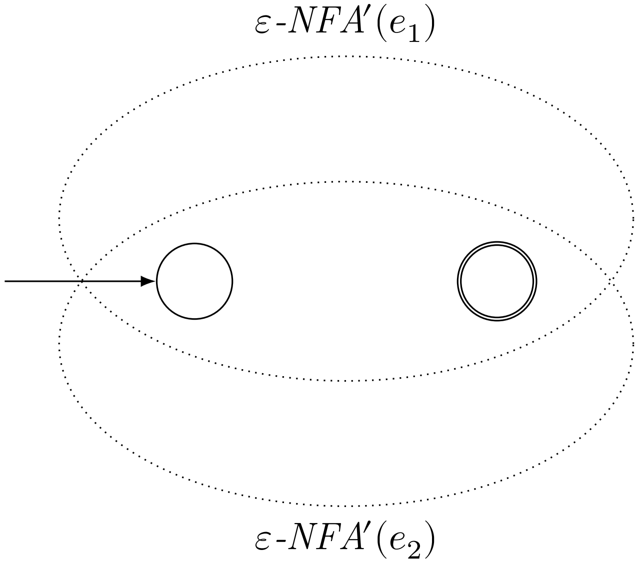
Alternation

$\varepsilon\text{-}NFA(e_1 + e_2) =$



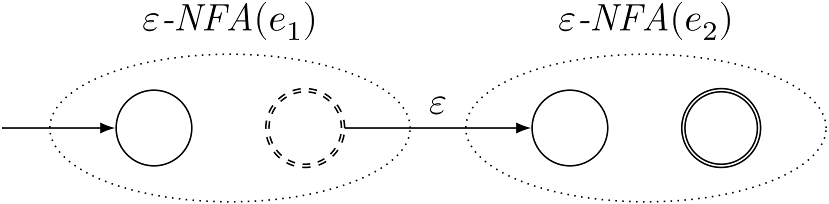
Alternation

$\varepsilon\text{-}NFA'(e_1 + e_2) =$



Sequencing

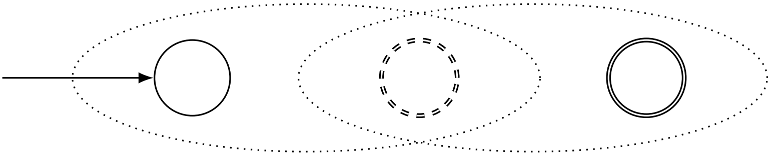
$\varepsilon\text{-}NFA(e_1e_2) =$



Sequencing

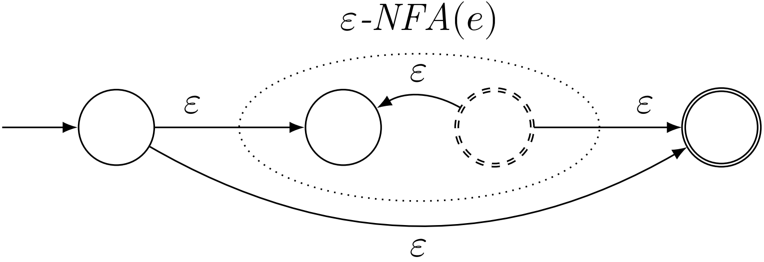
$$\varepsilon\text{-}NFA'(e_1 e_2) =$$

$$\varepsilon\text{-}NFA'(e_1) \quad \varepsilon\text{-}NFA'(e_2)$$



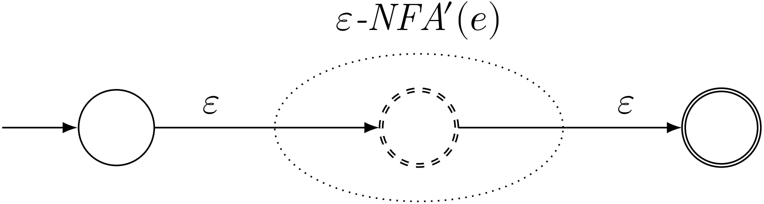
Kleene star

$\varepsilon\text{-}NFA(e^*) =$

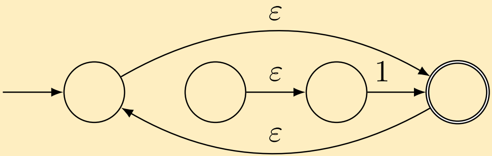


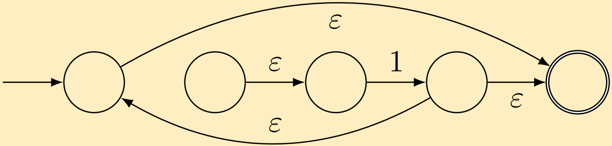
Kleene star

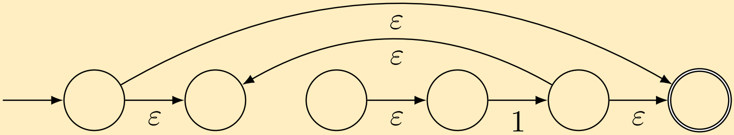
$$\varepsilon\text{-}NFA'(e^*) =$$



Which of the following ε-NFAs is equal to $\varepsilon\text{-}NFA((\emptyset 1)^*)$ (ignoring the alphabet and the names of the states)?

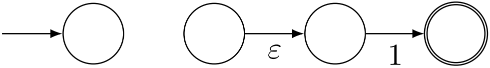
1. 

2. 

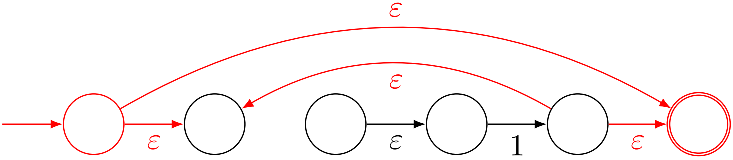
3. 

Respond at `https://pingo.coactum.de/729558`.

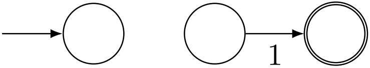
$\varepsilon\text{-}NFA(\emptyset 1) =$



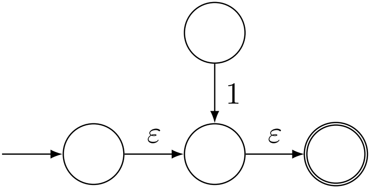
$\varepsilon\text{-}NFA((\emptyset 1)^*) =$



$\varepsilon\text{-}NFA'(\emptyset 1) =$



$\varepsilon\text{-}NFA'((\emptyset 1)^*) =$



Regular languages

- Recall that a language $M \subseteq \Sigma^*$ is regular if there is some DFA A with alphabet Σ such that $L(A) = M$.
- A language $M \subseteq \Sigma^*$ is also regular if and only if there is some *regular expression* $e \in RE(\Sigma)$ such that $L(e) = M$.

More about regular expression algebra

Discovering and proving laws

- In the last lecture I mentioned that $(\varepsilon + e)^* = e^*$.
- How can you figure out that this holds?
- And how can you prove it?

Proving laws

- Recall that $e_1 = e_2$ means that $L(e_1) = L(e_2)$.
- We can prove $L(e_1) = L(e_2)$ by proving $L(e_1) \subseteq L(e_2)$ and $L(e_2) \subseteq L(e_1)$, i.e. that $\forall w \in L(e_1).\ w \in L(e_2)$ and $\forall w \in L(e_2).\ w \in L(e_1)$.

Let $e \in RE(\Sigma)$. Then $(\varepsilon + e)^* = e^*$.

$L((\varepsilon + e)^*) \subseteq L(e^*)$:

- If $w \in L((\varepsilon + e)^*)$, then there is some $n \in \mathbb{N}$ such that $w = w_1 \cdots w_n$ and each string w_i is either ε or a member of $L(e)$.
- Remove the strings w_i that are equal to ε.
- We get a string $w' = w_{k_1} \cdots w_{k_m}$, for some natural numbers m and $k_1 < \cdots < k_m$.
- Because all strings w_{k_i} belong to $L(e)$ we get that $w' \in L(e^*)$.
- Furthermore $w = w'$, so $w \in L(e^*)$.

Let $e \in RE(\Sigma)$. Then $(\varepsilon + e)^* = e^*$.

$L(e^*) \subseteq L((\varepsilon + e)^*)$:

- We have that $L(e) \subseteq L(\varepsilon) \cup L(e) = L(\varepsilon + e)$.
- The result follows by monotonicity of $-^*$.

Monotonicity

- $M \subseteq N$ implies that $M^* \subseteq N^*$.
- $M_1 \subseteq N_1$ and $M_2 \subseteq N_2$ imply:
  - $M_1 \cup M_2 \subseteq N_1 \cup N_2$.
  - $M_1 \cap M_2 \subseteq N_1 \cap N_2$.
  - $M_1 M_2 \subseteq N_1 N_2$.

Discovering (and proving) laws

- A regular expression proposition $e_1 = e_2$ is valid iff the equation obtained by replacing each variable e by a *fresh* symbol a is true.
- Examples:
  - $(\varepsilon + e)^* = e^*$ is valid iff $(\varepsilon + 1)^* = 1^*$ is true.
  - $e_1 1 e_2 = e_2 1 e_1$ is valid iff $012 = 210$ is true.
- Next lecture: An algorithm for checking if two regular languages are equal.

Discovering (and proving) laws

- This "trick" is rather syntactic.
- It does not work if we include intersection among the regular expression operators. The proposition

$$L \cap M = L \cap N$$

is not valid, but

$$\{ a \} \cap \{ b \} = \{ a \} \cap \{ c \}$$

is true.
- One can construct similar counterexamples for $\overline{-}$ and $- \setminus -$.

Which of the following regular expression equivalences are valid?

1. $\emptyset^* e = e$.
2. $(e_1 + e_2)^* = e_1^* + (e_1 e_2)^* + e_2^*$.
3. $e_1(e_2 e_1)^* = (e_1 e_2)^* e_1$.
4. $(e_1 + e_2)^* = (e_1^* e_2)^* e_1^*$.
5. $(e_1 + e_2)^* = e_1^*(e_2 e_1^* e_2)^* e_1^*$.

Respond at `https://pingo.coactum.de/729558`.

The shifting and denesting rules

1. The shifting rule: $e_1(e_2e_1)^* = (e_1e_2)^*e_1$.
2. The denesting rule: $(e_1 + e_2)^* = (e_1^*e_2)^*e_1^*$.

The denesting rule

Consider the following equations:

$$e_1 = e_2$$
$$e_2 = 0e_1 + 1e_2 + \varepsilon$$

One way to find a solution for e_1, using Arden's lemma:

$$e_2 = (0+1)e_2 + \varepsilon$$
$$e_2 = (0+1)^*\varepsilon = (0+1)^*$$
$$e_1 = (0+1)^*$$

Another way:

$$e_2 = 1^*(0e_1 + \varepsilon)$$
$$e_1 = 1^*0e_1 + 1^*$$
$$e_1 = (1^*0)^*1^*$$

One can combine methods

Is it the case that $((\varepsilon + e)^*)^* \subseteq (1 + e)^*$?

- We know that $(\varepsilon + e)^* = e^*$, so $((\varepsilon + e)^*)^* = (e^*)^*$.
- We also have $e \subseteq 1 + e$, and thus, by monotonicity, $e^* \subseteq (1 + e)^*$.
- We can conclude if $(e^*)^* = e^*$.
- This holds if $(1^*)^* = 1^*$.
- We have $1^* \subseteq (1^*)^*$.
- We also have $(1^*)^* \subseteq 1^*$, because a string in $(1^*)^*$ consists of an arbitrary number of 1s, and is thus a member of 1^*.

More laws related to the Kleene star

1. $e^* = \varepsilon + ee^*$.
2. $e^*e^* = e^*$.
3. $(e^*)^* = e^*$.

The pumping lemma

The pumping lemma for regular languages

For every regular language L
over the alphabet Σ:

$$
\begin{aligned}
&\exists m \in \mathbb{N}.\\
&\quad \forall w \in L.\ |w| \geq m \Rightarrow\\
&\qquad \exists t, u, v \in \Sigma^*.\\
&\qquad\quad w = tuv \land |tu| \leq m \land u \neq \varepsilon \land\\
&\qquad\quad \forall n \in \mathbb{N}.\ tu^n v \in L
\end{aligned}
$$

The pumping lemma for regular languages

For every regular language L
over the alphabet Σ:

$$
\begin{aligned}
&\exists m \in \mathbb{N}.\\
&\quad \forall w \in L.\ |w| \geq m \Rightarrow\\
&\qquad \exists t, u, v \in \Sigma^*.\\
&\qquad\quad w = tuv \wedge |tu| \leq m \wedge u \neq \varepsilon \wedge\\
&\qquad\quad \forall n \in \mathbb{N}.\ tu^n v \in L
\end{aligned}
$$

The pumping lemma for regular languages

Proof sketch:

- There is at least one DFA $A = (Q, \Sigma, \delta, q_0, F)$ such that $L(A) = L$.
- Let $m = |Q|$.
- If a string $w \in \Sigma^*$ with $|w| \geq |Q|$ is accepted by A, then, by the pigeonhole principle, $\hat{\delta}(q_0, w_1 \cdots w_i) = \hat{\delta}(q_0, w_1 \cdots w_j)$ for some $i, j \in \{0, \ldots, |Q|\}$, $i < j$.
- Let $t = w_1 \cdots w_i$, $u = w_{i+1} \cdots w_j$, $v = w_{j+1} \cdots w_{|w|}$.
- Note that $tuv = w$, $|tu| \leq |Q|$ and $u \neq \varepsilon$.
- Furthermore $tv \in L$, $tu^2v \in L$, $tu^3v \in L$, ...

An application of the pumping lemma

New notation:

- w^{R}: The string w with the elements in reverse order.

Is the language $\{\, ww^{\mathsf{R}} \mid w \in \Sigma^* \,\}$ regular?

- It is if $|\Sigma| = 1$.
- But not if $|\Sigma| \geq 2$. We can prove this using the pumping lemma.

An application of the pumping lemma

- For simplicity, let $\Sigma = \{ a, b \}$.
- Denote the language by L:

$$L = \{ ww^{\mathsf{R}} \mid w \in \Sigma^* \}$$

An application of the pumping lemma

L is not regular:

- Assume that L is regular.
- By the pumping lemma there is some $m \in \mathbb{N}$ such that, for all $w \in L$ for which $|w| \geq m$, there are strings $t, u, v \in \Sigma^*$ such that $w = tuv$, $|tu| \leq m$, $u \neq \varepsilon$ and, for all $n \in \mathbb{N}$, $tu^n v \in L$.
- Let w be the string $a^m b^{2m} a^m$.
- Note that $w \in L$ and $|w| \geq m$.

An application of the pumping lemma

- We get that there are strings $t, u, v \in \Sigma^*$ such that $a^m b^{2m} a^m = tuv$, $|tu| \leq m$, $u \neq \varepsilon$ and, for all $n \in \mathbb{N}$, $tu^n v \in L$.
- Because $a^m b^{2m} a^m = tuv$ and $|tu| \leq m$ we know that u consists only of a's, and because $u \neq \varepsilon$ we know that u consists of at least one a.
- We also know that $tv \in L$. However, this is contradictory, because $tv = a^n b^{2m} a^m$ for some $n < m$.

Is the language $P \subseteq \{\, (,) \,\}^*$ regular?

$$\frac{}{\varepsilon \in P} \qquad\qquad \frac{w \in P}{(w) \in P}$$

1. Yes.
2. No.

Respond at `https://pingo.coactum.de/729558`.

Necessary, not sufficient

- I have seen students try to use the pumping lemma to prove that a language *is* regular.
- However, there are non-regular languages that satisfy the pumping lemma's formula ("$\exists m \in \mathbb{N}.$ ...").

Closure properties

Closure properties

Let $M, N \subseteq \Sigma^*$ be regular languages. Then

- M^* is regular,
- MN is regular,
- $M \cup N$ is regular,
- $M \cap N$ is regular,
- $\Sigma^* \setminus N$ is regular, and
- $M \setminus N$ is regular.
  (Note that $M \setminus N = M \cap (\Sigma^* \setminus N)$.)

For which of the following definitions of M is $M \setminus \{ 1^n \mid n \in \mathbb{N}, n > 0 \}$ regular?

1. $M = \{ 1 \} \cup L((21)^*)$.
2. $$\frac{}{\varepsilon \in M} \qquad \frac{w \in M}{1w2 \in M}$$
3. $$\frac{}{\varepsilon \in M} \qquad \frac{w \in M}{1w1 \in M}$$

Respond at `https://pingo.coactum.de/729558`.

Today

- Translation from regular expressions to finite automata.
- More about regular expression algebra.
- The pumping lemma for regular languages.
- Some closure properties for regular languages.

Next lecture

- Various algorithms.
- Equivalence of states.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-02-08

Today

- Various algorithms.
- Equivalence of states.

Some old algorithms

Some algorithms we have already seen

- (ε-)NFA to DFA. (Can be slow.)
- DFA to (ε-)NFA. (Fast.)
- FA to RE. (Can be slow.)
- RE to ε-NFA. (Fast.)

Empty?

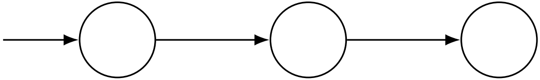
Is the language empty?

- For an FA: If there is no path from the start state to an accepting state.
- For a regular expression:

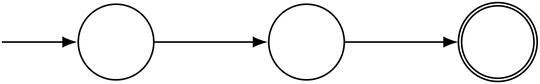
$$
\begin{aligned}
&empty \in RE(\Sigma) \to Bool\\
&empty(\emptyset) &&= \mathsf{true}\\
&empty(\varepsilon) &&= \mathsf{false}\\
&empty(a) &&= \mathsf{false}\\
&empty(e_1 e_2) &&= empty(e_1) \vee empty(e_2)\\
&empty(e_1 + e_2) &&= empty(e_1) \wedge empty(e_2)\\
&empty(e^*) &&= \mathsf{false}
\end{aligned}
$$

Is the language empty?

- Empty:



- Not empty:



Is the language empty?

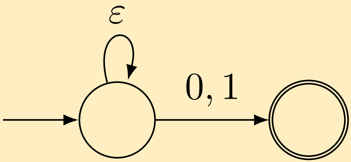
- Empty:

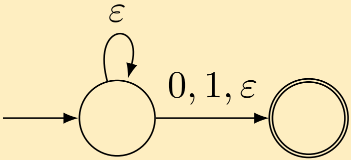
$$\begin{aligned} & empty(\emptyset 0^*) &= \\ & empty(\emptyset) \vee empty(0^*) &= \\ & \textsf{true} \vee \textsf{false} &= \\ & \textsf{true} \end{aligned}$$

- Not empty:

$$\begin{aligned} & empty(\emptyset + 0^*) &= \\ & empty(\emptyset) \wedge empty(0^*) &= \\ & \textsf{true} \wedge \textsf{false} &= \\ & \textsf{false} \end{aligned}$$

Which of the following regular expressions/ε-NFAs over $\{0, 1\}$ represent the empty language?

1. $\emptyset + \varepsilon$
2. $\emptyset + \emptyset^*$
3. $\emptyset^+$
4. $(\emptyset 01 + 10(\emptyset + \varepsilon\emptyset))^+$
5. 

6. 

Respond at `https://pingo.coactum.de/729558`.

Member?

Is the string a member of the language?

- For a DFA: Move from state to state, check if the last state is accepting.
- For an NFA or ε-NFA:
  - Keep track of a set of states.
  - Or convert to a DFA. (This could be much less efficient.)
- For a regular expression: Convert to an ε-NFA.

Equivalence of states

Equivalence of states

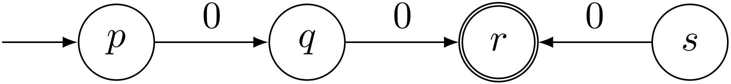
For a DFA $(Q, \Sigma, \delta, q_0, F)$:

- Two states $p, r \in Q$ are *equivalent* ($p \sim r$) if

$$\forall w \in \Sigma^*.\ \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(r, w) \in F.$$

- Two states that are not equivalent are *distinguishable*.

Equivalence of states



- The state p is equivalent to p.
- The state q is equivalent to q and s.
- The state r is equivalent to r.
- The state s is equivalent to q and s.

Which of the following properties does the $\sim$ relation always satisfy?

1. It is reflexive.
2. It is symmetric.
3. It is antisymmetric.
4. It is transitive.

Respond at `https://pingo.coactum.de/729558`.

Equivalence of states

To find out which states are equivalent:

- Create a matrix where rows and columns are labelled by states:

	s_0	s_1	s_2	s_3
s_0				
s_1				
s_2				
s_3				

Equivalence of states

To find out which states are equivalent:

- Create a matrix where rows and columns are labelled by states.
- Mark every accepting state as distinguishable from every non-accepting state.
- Repeat until no further changes are possible:
  - Mark two states $p, q \in Q$ as distinguishable if there is some $a \in \Sigma$ for which $\delta(p, a)$ and $\delta(q, a)$ have already been marked as distinguishable.
- States that have not been marked as distinguishable are equivalent.

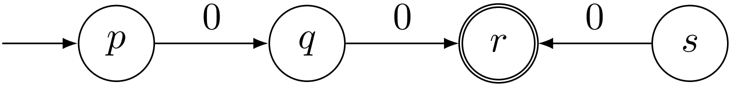
Equivalence of states

If row and column labels are ordered in the same way:

- The $\sim$ relation is reflexive, so one can skip the diagonal.
- The $\sim$ relation is symmetric, so one can skip, say, the elements below the diagonal.

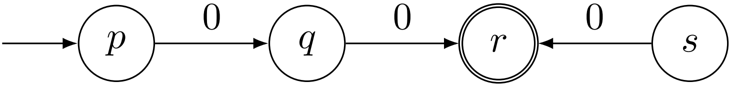
	s_0	s_1	s_2	s_3
s_0	.			
s_1	.	.		
s_2	.	.	.	
s_3	.	.	.	.

Equivalence of states



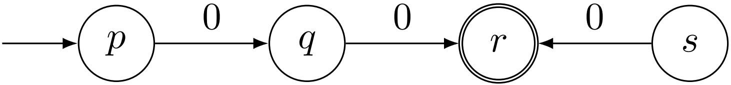
	p	q	r	s
p	.			
q	.	.		
r	.	.	.	
s	.	.	.	.

Equivalence of states



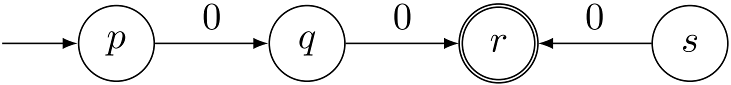
	p	q	r	s
p	.		x	
q	.	.	x	
r	.	.	.	x
s	.	.	.	.

Equivalence of states



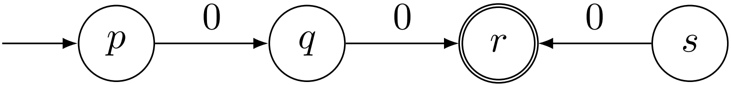
	p	q	r	s
p	.	x	x	
q	.	.	x	
r	.	.	.	x
s	.	.	.	.

Equivalence of states



	p	q	r	s
p	.	x	x	x
q	.	.	x	
r	.	.	.	x
s	.	.	.	.

Equivalence of states

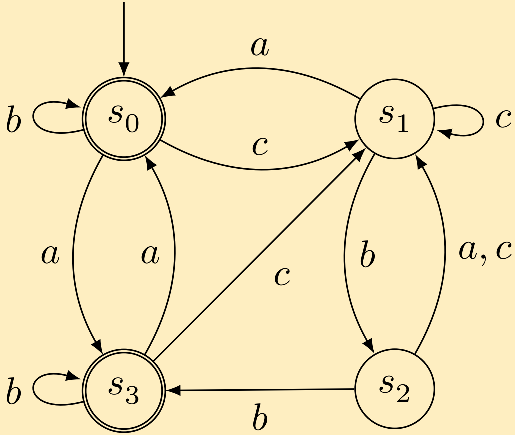


	p	q	r	s
p	.	x	x	x
q	.	.	x	
r	.	.	.	x
s	.	.	.	.

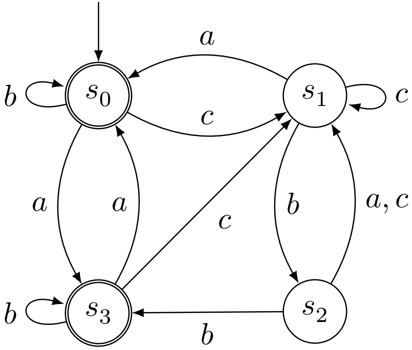
Equivalence of states

- The $\sim$ relation is an equivalence relation.
- The equivalence classes partition the set of states.

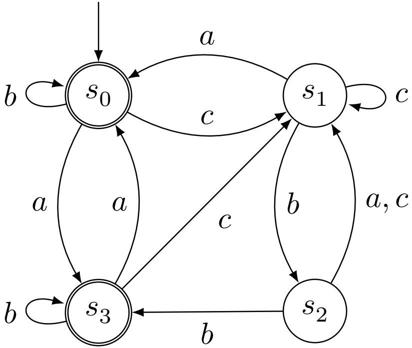
How many equivalence classes does the $\sim$ relation for the following DFA have?



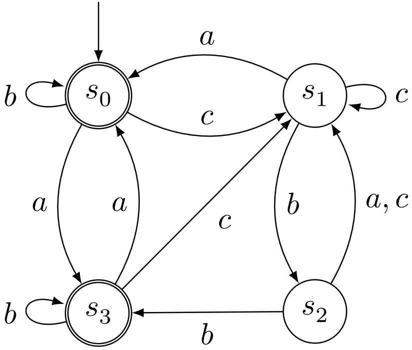
Respond at `https://pingo.coactum.de/729558`.



	s_0	s_1	s_2	s_3
s_0	.			
s_1	.	.		
s_2	.	.	.	
s_3	.	.	.	.



	s_0	s_1	s_2	s_3
s_0	.	X	X	
s_1	.	.		X
s_2	.	.	.	X
s_3	.	.	.	.



	s_0	s_1	s_2	s_3
s_0	.	x	x	
s_1	.	.	x	x
s_2	.	.	.	x
s_3	.	.	.	.

Equality of languages

Equality of languages

To find out if two languages, represented by the DFAs $(Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $(Q_2, \Sigma, \delta_2, q_{02}, F_2)$ with $Q_1 \cap Q_2 = \emptyset$, are equal:

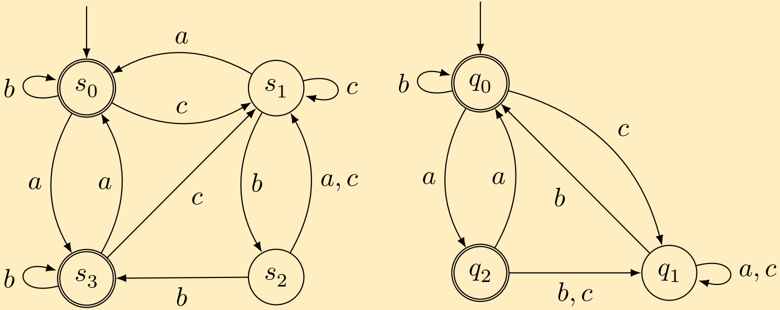
- Create the DFA $(Q_1 \cup Q_2, \Sigma, \delta, q_{01}, F_1 \cup F_2)$, where $\delta(q) = \delta_i(q)$ for $q \in Q_i$.
- The languages are equal iff $q_{01} \sim q_{02}$.

Equality of languages

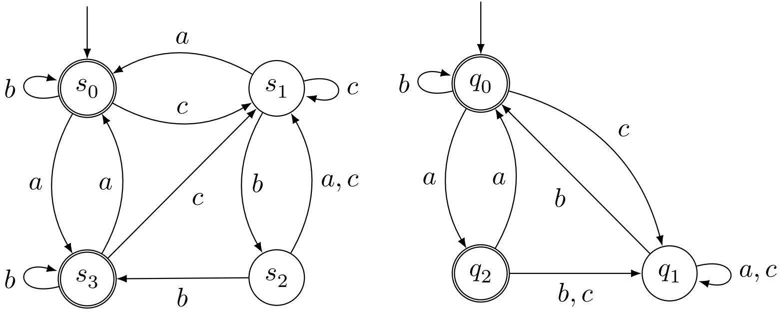
Note:

- If the "matrix method" above is used to decide whether $q_{01} \sim q_{02}$, then one can skip entries for which the row label and column label belong to the same DFA.

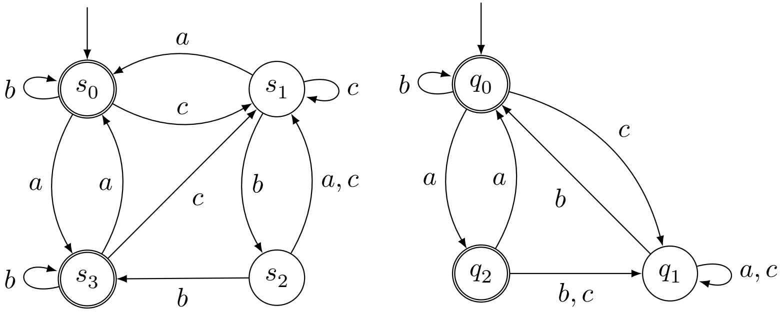
Are the languages over $\{ a, b, c \}$ denoted by the following DFAs equal?



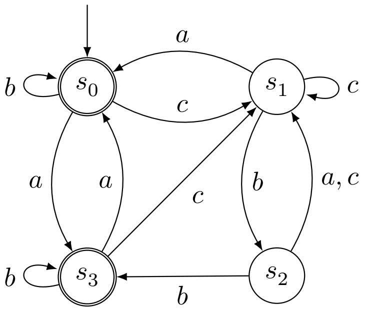
Respond at `https://pingo.coactum.de/729558`.

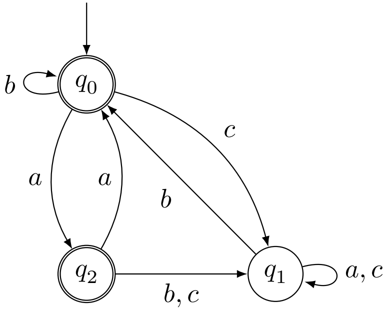


	q_0	q_1	q_2
s_0		x	
s_1	x		x
s_2	x		x
s_3		x	

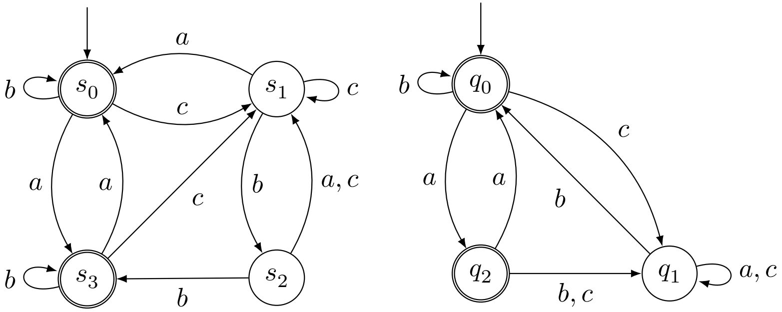


	q_0	q_1	q_2
s_0		x	
s_1	x	x	x
s_2	x		x
s_3		x	





	q_0	q_1	q_2
s_0	x	x	
s_1	x	x	x
s_2	x		x
s_3		x	



	q_0	q_1	q_2
s_0	x	x	x
s_1	x	x	x
s_2	x	x	x
s_3	x	x	x

Minimisation

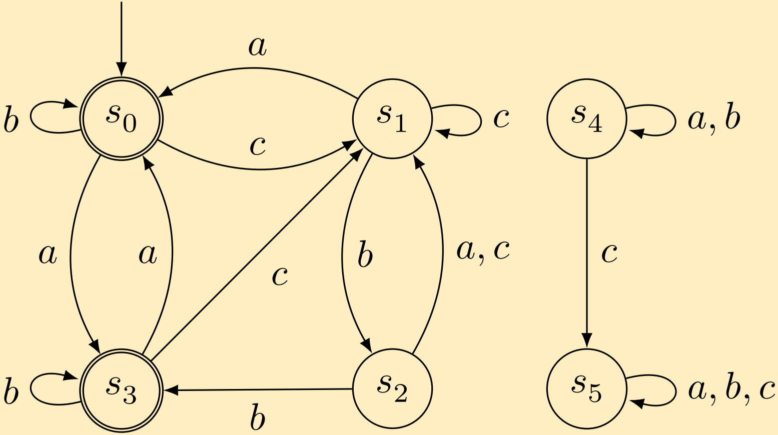
Minimisation

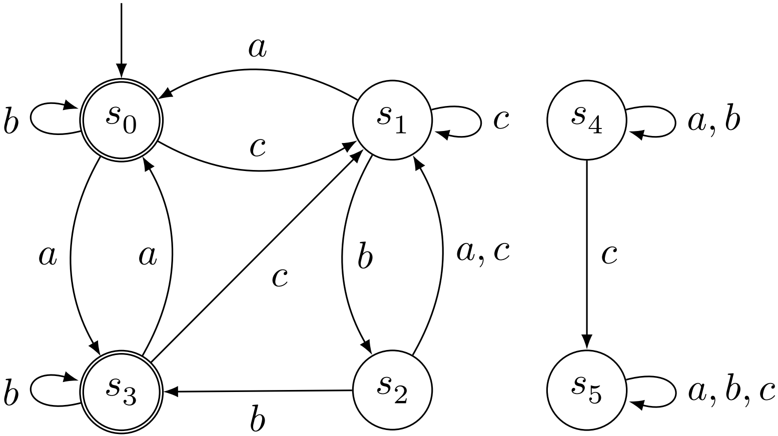
Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ one can construct a minimal (in terms of the number of states) DFA that represents the same language.

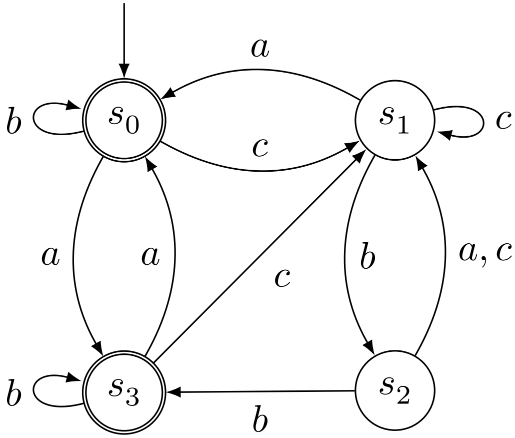
Minimisation

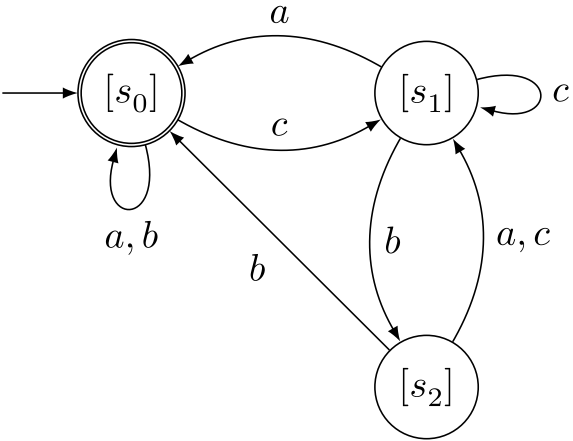
1. Remove non-accessible states.
2. Merge equivalent states.

Minimise the following DFA.




s_0
s_1
s_2
s_3
s_4
s_5
a
b
c
a, c
a, b
a, b, c


s_0
s_1
s_2
s_3
a
b
c
a
a
c
b
a, c
b
b
c


a
$[s_0]$
c
$[s_1]$
c
a, b
b
b
a, c
$[s_2]$

Minimisation

1. Remove non-accessible states:

$$A' = (Acc(q_0), \Sigma, \delta', q_0, F \cap Acc(q_0))$$
$$\delta'(q, a) = \delta(q, a)$$

2. Replace the set of states with equivalence classes of equivalent states:

$$A'' = (Acc(q_0)/\sim, \Sigma, \delta'', [q_0], F'')$$
$$\delta''([q], a) = [\delta(q, a)]$$
$$F'' = \{ \, [q] \mid q \in F \cap Acc(q_0) \, \}$$

Exercise: Check that A'' is a well-formed DFA. Prove that it accepts the same language as A.

Minimisation

Why is the constructed DFA minimal?

- Take any DFA $B = (Q_B, \Sigma, \delta_B, q_B, F_B)$ that represents the same language.
- Combine A'' and B like in the language equality checking algorithm (renaming states if necessary).
- We have $[q_0] \sim q_B$.
- Hence every accessible state $\widehat{\delta''}([q_0], w)$ of A'' is equivalent to a state of B, $\widehat{\delta_B}(q_B, w)$.

Minimisation

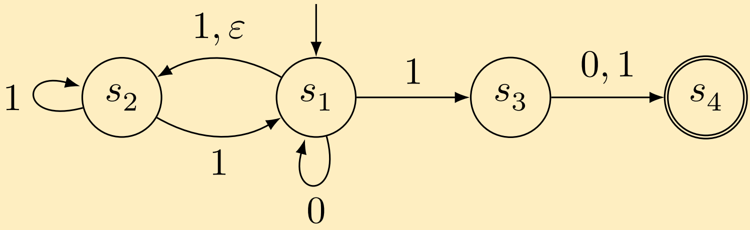
Why is the constructed DFA minimal?

- Every accessible state of A'' is equivalent to a state of B.
- Note that every state of A'' is accessible, so every state of A'' is equivalent to some state of B.
- Furthermore states of A'' that are not equal are not equivalent, so the equivalence classes of $\sim$ for the combined automaton contain at most one state from A'' and at least one state from B.
- Thus Q_B is at least as large as $Acc(q_0)/\sim$.

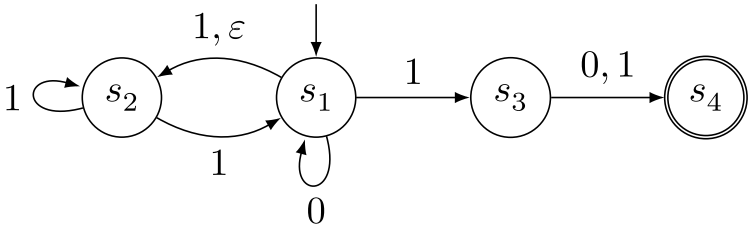
Minimisation

In fact, the minimised DFA is equal
(up to renaming of states)
to every other minimal DFA for the same language.

Consider the following ε-NFA over $\{ 0, 1 \}$. How many states does a minimal ε-NFA for the same language have? (Count only the number of states, not the number of edges.)



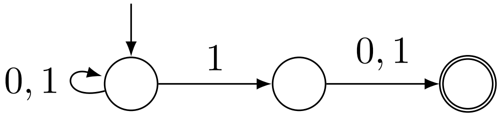
Respond at `https://pingo.coactum.de/729558`.



If we define $p \sim q$ by

$$\forall w \in \Sigma^*.\ \hat{\delta}(p, w) \cap F = \emptyset \ \Leftrightarrow\ \hat{\delta}(q, w) \cap F = \emptyset,$$

then we see that $\sim$ has four equivalence classes. However, the ε-NFA is not minimal:



Today

- Is the language empty?
- Is the string a member of the language?
- Equivalence of states.
- Are the languages equal?
- Minimisation of DFAs.

Next lecture

- Context-free grammars.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-02-12

Today

Context-free grammars:

- Syntax.
- Semantics.

Context-free grammars

Context-free grammars (CFGs)

- The context-free languages are those that can be described by CFGs.
- Every regular language is context-free.
- Some context-free languages are not regular.
- CFGs are for instance used to specify the syntax of some programming languages.
  - One example: Haskell.
- Parser generators often use (restricted) CFGs.

Syntax

Context-free grammars

A context-free grammar has the form (N, Σ, P, S):

- N is a finite set of *nonterminals*.
- Σ is a finite set of *terminals* satisfying $\Sigma \cap N = \emptyset$.
- $P \subseteq N \times (N \cup \Sigma)^*$ is a finite set of *productions*.
- The start symbol $S \in N$.

Notation

- A production (A, α) can be written $A \to \alpha$.
- Multiple productions $A \to \alpha_1$, …, $A \to \alpha_n$ can be written $A \to \alpha_1 \mid \cdots \mid \alpha_n$ (if $n \geq 2$).

Which of the following expressions are well-formed context-free grammars?

1. $(\mathbb{N}, \{ a, b \}, P, 0)$, where P contains the following productions: $0 \to a1$, $1 \to b$.
2. $(\{ 0, 1 \}, \{ a, b \}, P, 0)$, where P contains the following productions: $0 \to a1$, $1 \to b$.
3. $(\{ 0, 1 \}, \{ 0, 1 \}, P, 0)$, where P contains the following productions: $0 \to 01$, $1 \to 1$.
4. $(\{ 0, 1 \}, \{ 0', 1' \}, P, 0)$, where P contains the following productions: $0 \to 01$, $1 \to 1 \mid 0$.
5. $(\{ 0, 1 \}, \{ 0', 1' \}, P, 2)$, where P contains the following productions: $0 \to 01$, $1 \to 1 \mid 0$.

Respond at `https://pingo.coactum.de/729558`.

Examples

An example

A context-free grammar for the non-regular language $\{ 0^n 1^n \mid n \in \mathbb{N} \}$ over $\{ 0, 1 \}$:

$$(\{ S \}, \{ 0, 1 \}, S \to 0S1 \mid \varepsilon, S)$$

An example

A context-free grammar for the non-regular language $\{ 0^n 1^n \mid n \in \mathbb{N} \}$ over $\{ 0, 1 \}$:

$$(\{ S \}, \{ 0, 1 \}, S \to 0S1 \mid \varepsilon, S)$$

Generated strings:

- ε.
- $0\varepsilon 1 = 01$.
- 0011.
- $\vdots$

An example

A context-free grammar for the non-regular language $\{\, 0^n 1^n \mid n \in \mathbb{N} \,\}$ over $\{\, 0, 1 \,\}$:

$$(\{\, S \,\}, \{\, 0, 1 \,\}, S \to 0S1 \mid \varepsilon, S)$$

An inductive definition of the language $L \subseteq \{\, 0, 1 \,\}^*$ generated by the grammar:

$$\frac{w \in L}{0w1 \in L} \qquad \frac{}{\varepsilon \in L}$$

Another example

Consider the grammar $(\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A1 \mid \varepsilon \qquad\qquad A \to 1A0 \mid S \mid \varepsilon$$

Another example

Consider the grammar $(\{\, S, A \,\}, \{\, 0, 1 \,\}, P, S)$, where P is defined in the following way:

$$S \to 0A1 \mid \varepsilon \qquad\qquad A \to 1A0 \mid S \mid \varepsilon$$

Sentential forms:

- S.
- ε.
- $0A1$.
- $01A01$.
- $01S01$.
- 0101.
- $\vdots$

Another example

Consider the grammar $(\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A1 \mid \varepsilon \qquad\qquad A \to 1A0 \mid S \mid \varepsilon$$

An inductive definition of the languages $L_S, L_A \subseteq \{ 0, 1 \}^*$ generated by S and A:

$$\frac{w \in L_A}{0w1 \in L_S} \qquad\qquad \frac{}{\varepsilon \in L_S}$$

$$\frac{w \in L_A}{1w0 \in L_A} \qquad\qquad \frac{w \in L_S}{w \in L_A} \qquad\qquad \frac{}{\varepsilon \in L_A}$$

Construct a context-free grammar for the language $\{ 0^{3n}1^{2n} \mid n \in \mathbb{N} \}$ over $\{ 0, 1 \}$ by filling in the missing part of the following definition.

$(\{ S \}, \{ 0, 1 \}, S \to ???, S)$

Respond at `https://pingo.coactum.de/729558`.

Semantics

Some conventions

Following the course text book:

- A, B, C: Nonterminals.
- a, b, c: Terminals.
- X, Y, Z: Nonterminals or terminals.
- u, v, w: Lists of terminals.
- α, β, γ: Lists of terminals and/or nonterminals.

Derivations

For the grammar $G = (N, \Sigma, P, S)$ one can define the following two binary relations on $(N \cup \Sigma)^*$ inductively:

$$\frac{\alpha, \beta \in (N \cup \Sigma)^* \qquad A \in N \qquad (A, \gamma) \in P}{\alpha A \beta \Rightarrow \alpha \gamma \beta}$$

$$\frac{}{\alpha \Rightarrow^* \alpha} \qquad\qquad \frac{\alpha \Rightarrow \beta \qquad \beta \Rightarrow^* \gamma}{\alpha \Rightarrow^* \gamma}$$

The language $L(G) = \{\, w \in \Sigma^* \mid S \Rightarrow^* w \,\}$.

Derivations

Consider the following grammar:

$$(\{\, S \,\}, \{\, 0, 1 \,\}, S \to 0S1 \mid \varepsilon, S)$$

Some derivations:

$$\begin{aligned} S &\Rightarrow 0S1 \\ 0S1 &\Rightarrow 01 \end{aligned}$$

$$\begin{aligned} S &\Rightarrow^* S \\ S &\Rightarrow^* 0S1 \\ S &\Rightarrow^* 01 \\ 0S1 &\Rightarrow^* 01 \end{aligned}$$

Leftmost derivations

A variant:

$$\frac{w \in \Sigma^* \qquad A \in N \qquad \alpha \in (N \cup \Sigma)^* \qquad (A, \beta) \in P}{wA\alpha \Rightarrow_{\mathsf{lm}} w\beta\alpha}$$

$$\frac{}{\alpha \Rightarrow^*_{\mathsf{lm}} \alpha} \qquad\qquad \frac{\alpha \Rightarrow_{\mathsf{lm}} \beta \qquad \beta \Rightarrow^*_{\mathsf{lm}} \gamma}{\alpha \Rightarrow^*_{\mathsf{lm}} \gamma}$$

Leftmost derivations

Consider the grammar $(\{ S, A, B \}, \{ a, b \}, P, S)$, where P is defined in the following way:

$$S \to AB \qquad A \to a \qquad B \to b$$

Some examples:

$$\begin{aligned} S &\Rightarrow_{\mathsf{lm}} AB \\ AB &\Rightarrow_{\mathsf{lm}} aB \\ aB &\Rightarrow_{\mathsf{lm}} ab \\ AB &\not\Rightarrow_{\mathsf{lm}} Ab \end{aligned}$$

$$\begin{aligned} S &\Rightarrow_{\mathsf{lm}}^{*} ab \\ AB &\Rightarrow_{\mathsf{lm}}^{*} ab \end{aligned}$$

Which of the following propositions are valid?

1. $A \Rightarrow^* \beta \quad \Leftrightarrow \quad A \Rightarrow^*_{\mathsf{lm}} \beta$
2. $A \Rightarrow^* w \quad \Leftrightarrow \quad A \Rightarrow^*_{\mathsf{lm}} w$

Respond at `https://pingo.coactum.de/729558`.

Which of the following propositions are valid?

1. $A \Rightarrow^* \beta \quad \Leftrightarrow \quad A \Rightarrow^*_{\mathsf{lm}} \beta$

Counterexample:

$$G = (\{ S, A, B, C \}, \emptyset, \{ S \to AB, B \to C \}, S)$$
$$S \Rightarrow AB \Rightarrow AC$$
$$\neg(S \Rightarrow^*_{\mathsf{lm}} AC)$$

A bug

The course text book states that

$$A \Rightarrow^* \beta \quad \Leftrightarrow \quad A \Rightarrow^*_{\mathsf{lm}} \beta$$

holds. Do not trust everything that you read.

Which of the following propositions are valid?

2. $A \Rightarrow^* w \quad \Leftrightarrow \quad A \Rightarrow^*_{\mathsf{lm}} w$

Consider the following grammar again:

$$G = (\{ S, A, B, C \}, \emptyset, \{ S \to AB, B \to C \}, S)$$

The derivation

$$S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$$

is not a leftmost derivation, but one can reorder it:

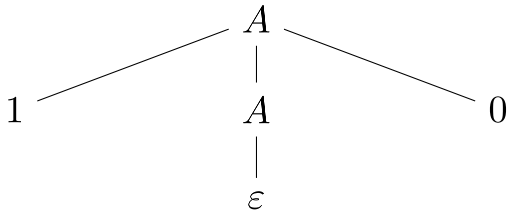
$$S \Rightarrow_{\mathsf{lm}} AB \Rightarrow_{\mathsf{lm}} aB \Rightarrow_{\mathsf{lm}} ab$$

Parse trees

Recall the grammar $(\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A1 \mid \varepsilon \qquad\qquad A \to 1A0 \mid S \mid \varepsilon$$

A parse tree:



The *yield* of this parse tree is the string 10.

Parse trees

An inductive definition of parse trees (for $G = (N, \Sigma, P, S)$):

- $P(G, A)$: Parse trees for G with the nonterminal $A \in N$ in the root.

$$\frac{(A, \alpha) \in P \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{node}(A, ts) \in P(G, A)}$$

Parse trees

An inductive definition of parse trees (for $G = (N, \Sigma, P, S)$):

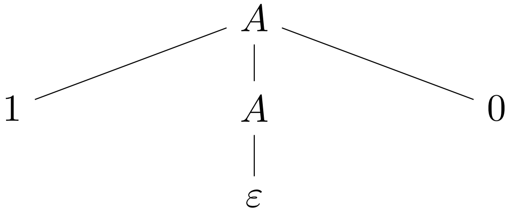
- $P_{\mathrm{L}}(G, \alpha)$: Lists of parse trees and terminals for G matching $\alpha \in (N \cup \Sigma)^*$.

$$\frac{}{\mathsf{nil} \in P_{\mathrm{L}}(G, \varepsilon)} \qquad \frac{a \in \Sigma \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{term}(a, ts) \in P_{\mathrm{L}}(G, a\alpha)}$$

$$\frac{A \in N \qquad t \in P(G, A) \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{nonterm}(t, ts) \in P_{\mathrm{L}}(G, A\alpha)}$$

Parse trees

Recall:



A corresponding parse tree in $P(G, A)$:

$$\mathsf{node}(A, \mathsf{term}(1, \mathsf{nonterm}(\mathsf{node}(A, \mathsf{nil}), \mathsf{term}(0, \mathsf{nil}))))$$

Parse trees

The yield of a parse tree (for $G = (N, \Sigma, P, S)$):

$$
\begin{aligned}
&\mathit{yield} \in P(G, A) \to \Sigma^* \\
&\mathit{yield}(\mathsf{node}(A, ts)) = \mathit{yield}_{\mathrm{L}}(ts)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{yield}_{\mathrm{L}} \in P_{\mathrm{L}}(G, \alpha) \to \Sigma^* \\
&\mathit{yield}_{\mathrm{L}}(\mathsf{nil}) &&= \varepsilon \\
&\mathit{yield}_{\mathrm{L}}(\mathsf{term}(a, ts)) &&= a\ \mathit{yield}_{\mathrm{L}}(ts) \\
&\mathit{yield}_{\mathrm{L}}(\mathsf{nonterm}(t, ts)) &&= \mathit{yield}(t)\ \mathit{yield}_{\mathrm{L}}(ts)
\end{aligned}
$$

Parse trees

The yield of a parse tree (for $G = (N, \Sigma, P, S)$):

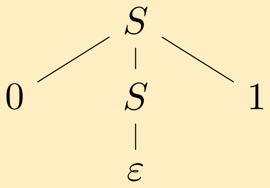
$$\begin{aligned}
&\mathit{yield} \in P(G, A) \to \Sigma^* \\
&\mathit{yield}(\mathsf{node}(A, ts)) = \mathit{yield}_{\mathrm{L}}(ts) \\
&\mathit{yield}_{\mathrm{L}} \in P_{\mathrm{L}}(G, \alpha) \to \Sigma^* \\
&\mathit{yield}_{\mathrm{L}}(\mathsf{nil}) && = \varepsilon \\
&\mathit{yield}_{\mathrm{L}}(\mathsf{term}(a, ts)) && = a\ \mathit{yield}_{\mathrm{L}}(ts) \\
&\mathit{yield}_{\mathrm{L}}(\mathsf{nonterm}(t, ts)) && = \mathit{yield}(t)\ \mathit{yield}_{\mathrm{L}}(ts)
\end{aligned}$$

- If $t \in P(G, S)$, then $\mathit{yield}(t) \in L(G)$.
- $L(G) = \{\, \mathit{yield}(t) \mid t \in P(G, S) \,\}$.

Consider the grammar
$G = (\{ S \}, \{ 0, 1 \}, S \to 0S1 \mid \varepsilon, S)$.

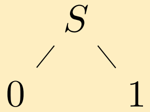
Which of the following statements hold for
$t = \mathsf{node}(S, \mathsf{term}(0, \mathsf{nonterm}(\mathsf{node}(S, \mathsf{nil}), \mathsf{term}(1, \mathsf{nil})))) \in P(G, S)$?

1. The following parse tree is equal to t:



2. $yield(t) = 0S1$

3. The following parse tree is equal to t:



4. $yield(t) = 01$

Recursive inference

If the grammar $G = (N, \Sigma, P, S)$, then one can define certain languages over Σ inductively:

- The language generated by the nonterminal $A \in N$, $L(G, A)$.
- The language generated by a list $\alpha \in (N \cup \Sigma)^*$, $L_{\mathrm{L}}(G, \alpha)$.

Recursive inference

If the grammar $G = (N, \Sigma, P, S)$, then one can define certain languages over Σ inductively:

- The language generated by the nonterminal $A \in N$, $L(G, A)$.
- The language generated by a list $\alpha \in (N \cup \Sigma)^*$, $L_{\mathrm{L}}(G, \alpha)$.

This is done in such a way that

- $L(G, A) = \{ \, \mathit{yield}(t) \mid t \in P(G, A) \, \}$ and
- $L_{\mathrm{L}}(G, \alpha) = \{ \, \mathit{yield}_{\mathrm{L}}(ts) \mid ts \in P_{\mathrm{L}}(G, \alpha) \, \}$.

Recursive inference

Consider the following definitions again:

$$\frac{(A, \alpha) \in P \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{node}(A, ts) \in P(G, A)}$$

$$\frac{}{\mathsf{nil} \in P_{\mathrm{L}}(G, \varepsilon)} \qquad \frac{a \in \Sigma \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{term}(a, ts) \in P_{\mathrm{L}}(G, a\alpha)}$$

$$\frac{A \in N \qquad t \in P(G, A) \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{nonterm}(t, ts) \in P_{\mathrm{L}}(G, A\alpha)}$$

Recursive inference

$$\frac{(A, \alpha) \in P \qquad w \in L_{\mathrm{L}}(G, \alpha)}{w \in L(G, A)}$$

$$\frac{}{\mathsf{nil} \in P_{\mathrm{L}}(G, \varepsilon)} \qquad \frac{a \in \Sigma \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{term}(a, ts) \in P_{\mathrm{L}}(G, a\alpha)}$$

$$\frac{A \in N \qquad t \in P(G, A) \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{nonterm}(t, ts) \in P_{\mathrm{L}}(G, A\alpha)}$$

Recursive inference

$$\frac{(A, \alpha) \in P \qquad w \in L_{\mathrm{L}}(G, \alpha)}{w \in L(G, A)}$$

$$\frac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)} \qquad \frac{a \in \Sigma \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{term}(a, ts) \in P_{\mathrm{L}}(G, a\alpha)}$$

$$\frac{A \in N \qquad t \in P(G, A) \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{nonterm}(t, ts) \in P_{\mathrm{L}}(G, A\alpha)}$$

Recursive inference

$$\frac{(A, \alpha) \in P \qquad w \in L_{\mathrm{L}}(G, \alpha)}{w \in L(G, A)}$$

$$\frac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)} \qquad \frac{a \in \Sigma \qquad w \in L_{\mathrm{L}}(G, \alpha)}{aw \in L_{\mathrm{L}}(G, a\alpha)}$$

$$\frac{A \in N \qquad t \in P(G, A) \qquad ts \in P_{\mathrm{L}}(G, \alpha)}{\mathsf{nonterm}(t, ts) \in P_{\mathrm{L}}(G, A\alpha)}$$

Recursive inference

Recursive inference:

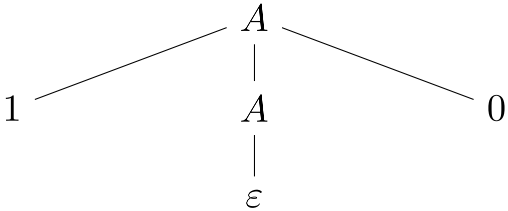
$$\frac{(A, \alpha) \in P \qquad w \in L_{\mathrm{L}}(G, \alpha)}{w \in L(G, A)}$$

$$\frac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)} \qquad \frac{a \in \Sigma \qquad w \in L_{\mathrm{L}}(G, \alpha)}{aw \in L_{\mathrm{L}}(G, a\alpha)}$$

$$\frac{A \in N \qquad v \in L(G, A) \qquad w \in L_{\mathrm{L}}(G, \alpha)}{vw \in L_{\mathrm{L}}(G, A\alpha)}$$

Recursive inference

Recall:



Recursive inference

Let us prove that $10 \in L(G, A)$:

$$\frac{A \to \alpha \in P \quad u \in L_{\mathrm{L}}(G, \alpha)}{u \in L(G, A)}$$

Recursive inference

Let us prove that $10 \in L(G, A)$:

$$\dfrac{\dfrac{}{A \to 1A0 \in P} \qquad u \in L_{\mathrm{L}}(G, 1A0)}{u \in L(G, A)}$$

Recursive inference

Let us prove that $10 \in L(G, A)$:

$$\dfrac{\dfrac{}{A \to 1A0 \in P} \quad \dfrac{v \in L_{\mathrm{L}}(G, A0)}{1v \in L_{\mathrm{L}}(G, 1A0)}}{1v \in L(G, A)}$$

I have omitted "$a \in \Sigma$".

Recursive inference

Let us prove that $10 \in L(G, A)$:

$$\dfrac{\dfrac{}{A \to 1A0 \in P} \qquad \dfrac{\dfrac{v_1 \in L(G, A) \quad v_2 \in L_{\mathrm{L}}(G, 0)}{v_1 v_2 \in L_{\mathrm{L}}(G, A0)}}{1 v_1 v_2 \in L_{\mathrm{L}}(G, 1A0)}}{1 v_1 v_2 \in L(G, A)}$$

I have omitted “$a \in \Sigma$” and “$A \in N$”.

Recursive inference

Let us prove that $10 \in L(G, A)$:

$$
\dfrac{
  \dfrac{}{A \to 1A0 \in P}
  \qquad
  \dfrac{
    \dfrac{
      v_1 \in L(G, A)
      \qquad
      \dfrac{
        \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}
      }{0 \in L_{\mathrm{L}}(G, 0)}
    }{v_1 0 \in L_{\mathrm{L}}(G, A0)}
  }{1 v_1 0 \in L_{\mathrm{L}}(G, 1A0)}
}{1 v_1 0 \in L(G, A)}
$$

I have omitted "$a \in \Sigma$" and "$A \in N$".

Recursive inference

Let us prove that $10 \in L(G, A)$:

$$
\dfrac{\dfrac{}{A \to 1A0 \in P} \qquad \dfrac{\dfrac{w \in L(G,A) \qquad \dfrac{\dfrac{}{\varepsilon \in L_{\mathrm{L}}(G,\varepsilon)}}{0 \in L_{\mathrm{L}}(G,0)}}{w0 \in L_{\mathrm{L}}(G, A0)}}{1w0 \in L_{\mathrm{L}}(G, 1A0)}}{1w0 \in L(G, A)}
$$

I have omitted "$a \in \Sigma$" and "$A \in N$".

Recursive inference

Let us prove that $10 \in L(G, A)$:

$$
\dfrac{
  \dfrac{}{A \to 1A0 \in P}
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{A \to \alpha \in P \quad w \in L_{\mathrm{L}}(G, \alpha)}{w \in L(G, A)}
      \qquad
      \dfrac{\dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{0 \in L_{\mathrm{L}}(G, 0)}
    }{w0 \in L_{\mathrm{L}}(G, A0)}
  }{1w0 \in L_{\mathrm{L}}(G, 1A0)}
}{1w0 \in L(G, A)}
$$

I have omitted "$a \in \Sigma$" and "$A \in N$".

Recursive inference

Let us prove that $10 \in L(G, A)$:

$$
\dfrac{
  \dfrac{}{A \rightarrow 1A0 \in P}
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{}{A \rightarrow \varepsilon \in P}
        \quad
        w \in L_{\mathrm{L}}(G, \varepsilon)
      }{
        w \in L(G, A)
      }
      \qquad
      \dfrac{
        \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}
      }{
        0 \in L_{\mathrm{L}}(G, 0)
      }
    }{
      w0 \in L_{\mathrm{L}}(G, A0)
    }
  }{
    1w0 \in L_{\mathrm{L}}(G, 1A0)
  }
}{
  1w0 \in L(G, A)
}
$$

I have omitted "$a \in \Sigma$" and "$A \in N$".

Recursive inference

Let us prove that $10 \in L(G, A)$:

$$
\frac{
  \dfrac{}{A \to 1A0 \in P}
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{}{A \to \varepsilon \in P}
        \quad
        \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}
      }{\varepsilon \in L(G, A)}
      \quad
      \dfrac{
        \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}
      }{0 \in L_{\mathrm{L}}(G, 0)}
    }{\varepsilon 0 \in L_{\mathrm{L}}(G, A0)}
  }{1\varepsilon 0 \in L_{\mathrm{L}}(G, 1A0)}
}{1\varepsilon 0 \in L(G, A)}
$$

I have omitted "$a \in \Sigma$" and "$A \in N$".

Recursive inference

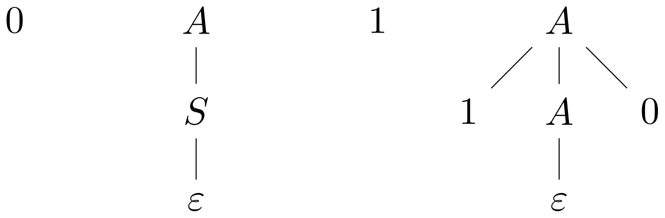
Let us prove that $10 \in L(G, A)$:

$$\dfrac{\dfrac{}{A \to 1A0 \in P} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{A \to \varepsilon \in P} \quad \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{\varepsilon \in L(G, A)} \quad \dfrac{\dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{0 \in L_{\mathrm{L}}(G, 0)}}{0 \in L_{\mathrm{L}}(G, A0)}}{10 \in L_{\mathrm{L}}(G, 1A0)}}{10 \in L(G, A)}$$

I have omitted "$a \in \Sigma$" and "$A \in N$".

Recursive inference

- A derivation of $w \in L_{\mathrm{L}}(G, \alpha)$ corresponds to a kind of parse forest.
- The forest corresponding to one derivation of $0110 \in L_{\mathrm{L}}(G, 0A1A)$:



Which of the following propositions are true? $G = (\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

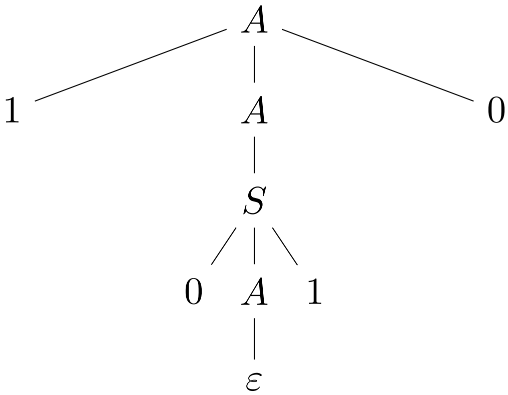
$$S \to 0A1 \mid \varepsilon \qquad A \to 1A0 \mid S \mid \varepsilon$$

1. $1010 \in L(G, S)$
2. $1010 \in L(G, A)$
3. $010 \in L_{\mathrm{L}}(G, S0A)$
4. $0100 \in L_{\mathrm{L}}(G, SA)$

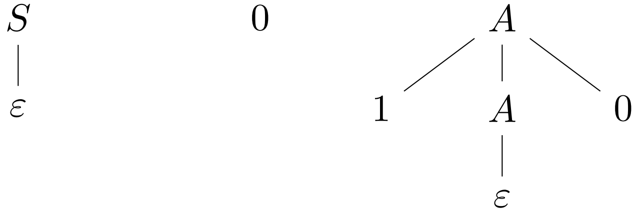
Hint: Try to construct parse trees.

Respond at `https://pingo.coactum.de/729558`.

$$S \to 0A1 \mid \varepsilon \qquad\qquad A \to 1A0 \mid S \mid \varepsilon$$



$$S \to 0A1 \mid \varepsilon \qquad\qquad A \to 1A0 \mid S \mid \varepsilon$$



$$
\dfrac{
  \dfrac{\dfrac{}{S \to \varepsilon \in P} \qquad \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{\varepsilon \in L(G, S)}
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{
        \dfrac{}{A \to 1A0 \in P}
        \qquad
        \dfrac{
          \dfrac{
            \dfrac{\dfrac{}{A \to \varepsilon \in P} \qquad \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{\varepsilon \in L(G, A)}
            \qquad
            \dfrac{\dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{0 \in L_{\mathrm{L}}(G, 0)}
          }{0 \in L_{\mathrm{L}}(G, A0)}
        }{10 \in L_{\mathrm{L}}(G, 1A0)}
      }{10 \in L(G, A)}
      \qquad
      \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}
    }{10 \in L_{\mathrm{L}}(G, A)}
  }{010 \in L_{\mathrm{L}}(G, 0A)}
}{010 \in L_{\mathrm{L}}(G, S0A)}
$$

Yields containing nonterminals

The inductive definitions of parse trees and recursive inference can be extended to support strings containing both terminals and nonterminals:

$$\frac{}{\mathsf{leaf}(A) \in P_{\mathrm{N}}(G, A)} \qquad \frac{}{A \in L_{\mathrm{N}}(G, A)}$$

$$\begin{aligned}
&\mathit{yield} \in P_{\mathrm{N}}(G, A) \to (N \cup \Sigma)^* \\
&\mathit{yield}(\mathsf{leaf}(A)) \qquad = A \\
&\mathit{yield}(\mathsf{node}(A, ts)) = \mathit{yield}_{\mathrm{L}}(ts) \\
&\mathit{yield}_{\mathrm{L}} \in P_{\mathrm{NL}}(G, \alpha) \to (N \cup \Sigma)^* \\
&\vdots
\end{aligned}$$

Proofs about grammars

A proof

Recall:

$$G = (\{ S \}, \{ 0, 1 \}, S \to 0S1 \mid \varepsilon, S)$$

$$\frac{w \in L}{0w1 \in L} \qquad\qquad \frac{}{\varepsilon \in L}$$

Let us prove that $L(G, S) \subseteq L$.

A proof

Let us prove $\forall u \in L(G, S).\ u \in L$ by complete induction on the length of the string. Assume that $u \in L(G, S)$. The derivation must end in the following way:

$$\frac{\dfrac{}{S \to \alpha \in P} \quad u \in L_{\mathrm{L}}(G, \alpha)}{u \in L(G, S)}$$

We have two cases, $\alpha = \varepsilon$ and $\alpha = 0S1$.

A proof

If $\alpha = \varepsilon$, then we have the following derivation:

$$\frac{\dfrac{}{S \to \varepsilon \in P} \qquad \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{\varepsilon \in L(G, S)}$$

We have $u = \varepsilon$, and $\varepsilon \in L$: $\dfrac{}{\varepsilon \in L}$.

A proof

If $\alpha = 0S1$, then the derivation ends in the following way:

$$\dfrac{\dfrac{}{S \to 0S1 \in P} \qquad \dfrac{\dfrac{w \in L(G,S) \qquad \dfrac{\dfrac{}{\varepsilon \in L_{\mathrm{L}}(G,\varepsilon)}}{1 \in L_{\mathrm{L}}(G,1)}}{w1 \in L_{\mathrm{L}}(G,S1)}}{0w1 \in L_{\mathrm{L}}(G,0S1)}}{0w1 \in L(G,S)}$$

- We have $u = 0w1$ for $w \in L(G,S)$.
- We also have $|w| < |u|$,
  so by the inductive hypothesis $w \in L$.
- Thus $u = 0w1 \in L$: $\dfrac{w \in L}{0w1 \in L}$.

A proof

- Another kind of induction can also be used: induction on the structure of the recursive inference.
- Exercise (optional, hard):
    - Write down a formula for this kind of induction.
    - Use this kind of induction to prove $L(G, S) \subseteq L$.

Another proof

- Let us now prove that $L \subseteq L(G, S)$.
- This is equivalent to $\forall w \in L.\ w \in L(G, S)$.
- Let us prove this by induction on the structure of L.

Another proof

- $\dfrac{}{\varepsilon \in L}$:

$$\dfrac{\dfrac{}{S \to \varepsilon \in P} \quad \dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{\varepsilon \in L(G, S)}$$

- $\dfrac{w \in L}{0w1 \in L}$:

$$\dfrac{\dfrac{}{S \to 0S1 \in P} \quad \dfrac{\dfrac{w \in L(G, S) \quad \dfrac{\dfrac{}{\varepsilon \in L_{\mathrm{L}}(G, \varepsilon)}}{1 \in L_{\mathrm{L}}(G, 1)}}{w1 \in L_{\mathrm{L}}(G, S1)}}{0w1 \in L_{\mathrm{L}}(G, 0S1)}}{0w1 \in L(G, S)}$$

Today

- Context-free grammars.
- Derivations.
- Left-most derivations.
- Parse trees.
- Recursive inference.
- Proofs about grammars.

Next lecture

- More about context-free grammars.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-02-19

Today

- Context-free languages.
- Some equivalences.
- Ambiguity.
- Designing grammars.

Context-free languages

Context-free languages

A language $L \subseteq \Sigma^*$ is context-free if $L = L(G)$, where G is a context-free grammar with Σ as the set of terminals.

Some equivalences

Some equivalences

With $w \in \Sigma^*$:

$$\begin{tikzcd}
 & A \Rightarrow^*_{\mathsf{lm}} w \arrow[dl, Rightarrow] & \\
A \Rightarrow^* w \arrow[rr, Rightarrow] & & w \in L_{\mathrm{N}}(G, A) \arrow[ul, Rightarrow] \arrow[dl, Rightarrow] \arrow[dd, Leftrightarrow] \\
 & A \Rightarrow^*_{\mathsf{rm}} w \arrow[ul, Rightarrow] & \\
 & & \begin{array}{l} \exists p \in P_{\mathrm{N}}(G, A). \\ \mathit{yield}(p) = w \end{array}
\end{tikzcd}$$

Some equivalences

With $w \in \Sigma^*$:

$$
\begin{tikzcd}
 & A \Rightarrow^*_{\mathsf{lm}} w \arrow[dl, Rightarrow, "\text{Easy}"] & \\
A \Rightarrow^* w \arrow[rr, Rightarrow] & & w \in L_{\mathrm{N}}(G, A) \arrow[ul, Rightarrow] \arrow[dl, Rightarrow] \arrow[dd, Leftrightarrow] \\
 & A \Rightarrow^*_{\mathsf{rm}} w \arrow[ul, Rightarrow, "\text{Easy}"] & \\
 & & \begin{array}{l} \exists p \in P_{\mathrm{N}}(G, A). \\ \mathit{yield}(p) = w \end{array}
\end{tikzcd}
$$

Some equivalences

With $w \in \Sigma^*$:

$$
\begin{tikzcd}
 & A \Rightarrow^*_{\mathsf{lm}} w \arrow[dl, Rightarrow] & \\
A \Rightarrow^* w \arrow[rr, Rightarrow] & & w \in L_{\mathrm{N}}(G, A) \arrow[ul, Rightarrow] \arrow[dl, Rightarrow] \arrow[dd, Leftrightarrow, "\text{Easy}"] \\
 & A \Rightarrow^*_{\mathsf{rm}} w \arrow[ul, Rightarrow] & \\
 & & \begin{array}{l} \exists p \in P_{\mathrm{N}}(G, A). \\ yield(p) = w \end{array}
\end{tikzcd}
$$

Some equivalences

With $w \in \Sigma^*$:

$$\begin{tikzcd}
 & A \Rightarrow^*_{\mathsf{lm}} w \arrow[dl, Rightarrow] & \\
A \Rightarrow^* w \arrow[rr, Rightarrow] & & w \in L_{\mathrm{N}}(G, A) \arrow[ul, Rightarrow] \arrow[dl, Rightarrow] \arrow[dd, Leftrightarrow] \\
 & A \Rightarrow^*_{\mathsf{rm}} w \arrow[ul, Rightarrow] & \\
 & & \begin{array}{l} \exists p \in P_{\mathrm{N}}(G, A). \\ \mathit{yield}(p) = w \end{array}
\end{tikzcd}$$

Some equivalences

With $\alpha \in (N \cup \Sigma)^*$:

$$
\begin{tikzcd}
 & A \Rightarrow^*_{\mathsf{lm}} \alpha \arrow[dl, Rightarrow] & \\
A \Rightarrow^* \alpha \arrow[rr, Leftrightarrow] & & \alpha \in L_{\mathrm{N}}(G, A) \arrow[dd, Leftrightarrow] \\
 & A \Rightarrow^*_{\mathsf{rm}} \alpha \arrow[ul, Rightarrow] & \\
 & & \begin{array}{l} \exists p \in P_{\mathrm{N}}(G, A). \\ yield(p) = \alpha \end{array}
\end{tikzcd}
$$

Some equivalences

One implication:

$$\forall A \in N, \beta \in (N \cup \Sigma)^*.$$
$$(A \Rightarrow^* \beta) \;\Rightarrow\; \beta \in L_{\mathrm{N}}(G, A)$$

Note that $\Rightarrow$ has two different meanings!

Some equivalences

One implication:

$$\forall A \in N, \beta \in (N \cup \Sigma)^*.$$
$$(A \Rightarrow^* \beta) \;\Rightarrow\; \beta \in L_{\mathrm{N}}(G, A)$$

One way to read this (in constructive logic):

- For all A in N and β in $(N \cup \Sigma)^*$...
- ...one can transform a proof of $A \Rightarrow^* \beta$ into a proof of $\beta \in L_{\mathrm{N}}(G, A)$.

Derivations as data

Derivations as data

Let us represent derivations as data in the following way:

- Let $Step = (\Sigma \cup N)^* \times N \times (\Sigma \cup N)^* \times (\Sigma \cup N)^*$.
- A single derivation step $\alpha A \beta \Rightarrow \alpha \gamma \beta$ is represented by the four-tuple $(\alpha, A, \gamma, \beta) \in Step$.
- A derivation is represented by a list of steps (in $List(Step)$) corresponding to the derivation steps.

Derivations as data

Examples:

- $S \Rightarrow \varepsilon$ is represented by $[(\varepsilon, S, \varepsilon, \varepsilon)]$.
- $S \Rightarrow 0S1 \Rightarrow 01$ is represented by $[(\varepsilon, S, 0S1, \varepsilon), (0, S, \varepsilon, 1)]$.
- $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$ is represented by $[(\varepsilon, S, 0S1, \varepsilon), (0, S, 0S1, 1), (00, S, \varepsilon, 11)]$.

Derivations as data

One can use the following function when constructing derivations:

$$
\begin{aligned}
& \mathit{wrap} \in (\Sigma \cup N)^* \times \mathit{List}(\mathit{Step}) \times (\Sigma \cup N)^* \to \\
& \qquad\qquad \mathit{List}(\mathit{Step}) \\
& \mathit{wrap}(\alpha', [\,], \qquad\qquad\qquad\qquad \beta') = [\,] \\
& \mathit{wrap}(\alpha', (\alpha, A, \gamma, \beta) : ss, \beta') = \\
& \quad (\alpha'\alpha, A, \gamma, \beta\beta') : \mathit{wrap}(\alpha', ss, \beta')
\end{aligned}
$$

Derivations as data

Examples:

- $S \Rightarrow \varepsilon$ is represented by $[(\varepsilon, S, \varepsilon, \varepsilon)]$.
- $S \Rightarrow 0S1 \Rightarrow 01$ starts with $S \Rightarrow 0S1$ and continues with $S \Rightarrow \varepsilon$ "wrapped" in 0 and 1, and is represented by

$$(\varepsilon, S, 0S1, \varepsilon) : wrap(0, [(\varepsilon, S, \varepsilon, \varepsilon)], 1) = [(\varepsilon, S, 0S1, \varepsilon), (0, S, \varepsilon, 1)].$$

Derivations as data

Examples:

- $S \Rightarrow \varepsilon$ is represented by $[(\varepsilon, S, \varepsilon, \varepsilon)]$.
- $S \Rightarrow 0S1 \Rightarrow 01$ starts with $S \Rightarrow 0S1$ and continues with $S \Rightarrow \varepsilon$ "wrapped" in 0 and 1, and is represented by

$$(\varepsilon, S, 0S1, \varepsilon) : \mathit{wrap}(0, [(\varepsilon, S, \varepsilon, \varepsilon)], 1) = [(\varepsilon, S, 0S1, \varepsilon), (0, S, \varepsilon, 1)].$$

- $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$ is represented by

$$(\varepsilon, S, 0S1, \varepsilon) : \mathit{wrap}(0, [(\varepsilon, S, 0S1, \varepsilon), (0, S, \varepsilon, 1)], 1).$$

Derivations as data

- Let $G = (\{ S \}, \{ 0, 1 \}, S \to 0S1 \mid \varepsilon, S)$.
- Note that $L(G) = \{ 0^n 1^n \mid n \in \mathbb{N} \}$.
- The function *derivation* takes a number $n \in \mathbb{N}$ to a derivation showing that $S \Rightarrow^* 0^n 1^n$:

$$
\begin{aligned}
&\mathit{derivation} \in \mathbb{N} \to \mathit{List}(\mathit{Step}) \\
&\mathit{derivation}(\mathsf{zero}) \quad\;\; = [(\varepsilon, S, \varepsilon, \varepsilon)] \\
&\mathit{derivation}(\mathsf{suc}(n)) = \\
&\quad (\varepsilon, S, 0S1, \varepsilon) : \mathit{wrap}(0, \mathit{derivation}(n), 1)
\end{aligned}
$$

Consider the grammar

$$G = (\{ S \}, \{ 0, 1 \}, S \rightarrow 1S \mid 0, S),$$

for which $L(G) = \{ 1^n 0 \mid n \in \mathbb{N} \}$.

Define a function that takes a number $n \in \mathbb{N}$ to a derivation showing that $S \Rightarrow^* 1^n 0$.

$$
\begin{aligned}
&derivation \in \mathbb{N} \to List(Step) \\
&derivation(\mathsf{zero}) \quad\;\; = [(\varepsilon, S, 0, \varepsilon)] \\
&derivation(\mathsf{suc}(n)) = \\
&\quad (\varepsilon, S, 1S, \varepsilon) \mathbin{:} wrap(1, derivation(n), \varepsilon)
\end{aligned}
$$

Ambiguity

Ambiguity

A grammar $G = (N, \Sigma, P, S)$ is ambiguous if there is a string $w \in \Sigma^*$ such that there are two different…

- …parse trees in $P(G, S)$ with yield w.
- …leftmost derivations $S \Rightarrow^*_{\mathsf{lm}} w$.
- …rightmost derivations $S \Rightarrow^*_{\mathsf{rm}} w$.
- …derivations of $w \in L(G, S)$.

Ambiguity

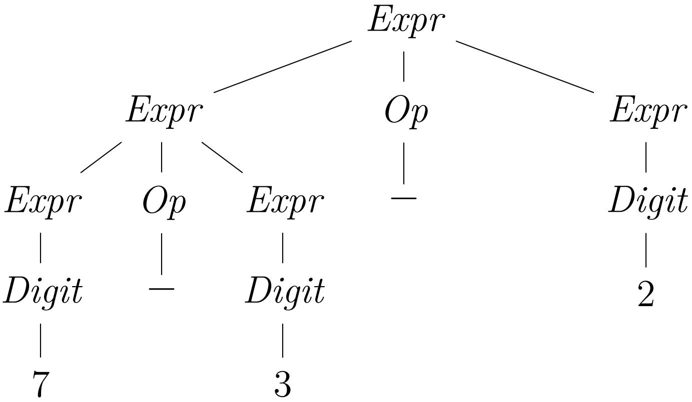
Consider the following (underspecified) context-free grammar over $\{+, -, \cdot, /, (,)\} \cup \{0, 1, ..., 9\}$:

$$
\begin{array}{lll}
\mathit{Expr} & \to & \mathit{Expr}\ \mathit{Op}\ \mathit{Expr} \mid \mathit{Digit} \mid (\ \mathit{Expr}\ ) \\
\mathit{Op} & \to & + \mid - \mid \cdot \mid / \\
\mathit{Digit} & \to & 0 \mid 1 \mid ... \mid 9
\end{array}
$$

How should $7 - 3 - 2$ be interpreted?

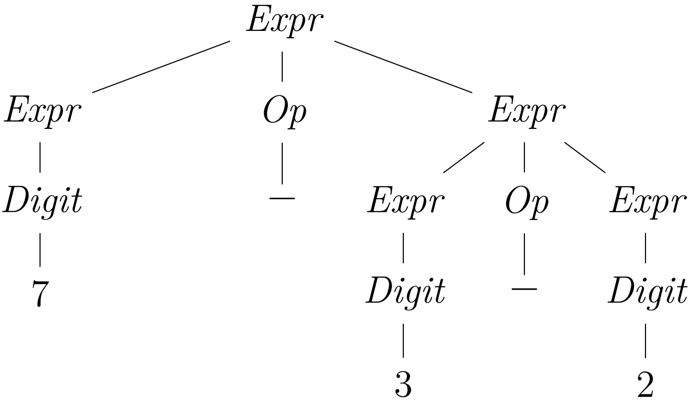
Ambiguity

A parse tree for $7 - 3 - 2$:



Ambiguity

Another parse tree for $7 - 3 - 2$:



Ambiguity

- The values differ: $(7-3)-2=2$, but $7-(3-2)=6$.
- If a grammar is used to determine how to interpret an expression, then it may be unclear how to interpret an ambiguous string.

For which of the following sets of productions P is $(\{ S, A \}, \{ 0, 1 \}, P, S)$ an ambiguous grammar?

1. $S \to S$
2. $S \to S \mid \varepsilon$
3. $S \to 1S1 \mid 0S0 \mid \varepsilon$
4. $S \to 1S1 \mid 1A1 \mid \varepsilon, A \to 1A1 \mid S$
5. $S \to 1S1 \mid 1A1 \mid \varepsilon, A \to 0S0$

Respond at `https://pingo.coactum.de/729558`.

For which of the following sets of productions P is $(\{ S, A \}, \{ 0, 1 \}, P, S)$ an ambiguous grammar?

1. $S \rightarrow S$
3. $S \rightarrow 1S1 \mid 0S0 \mid \varepsilon$
5. $S \rightarrow 1S1 \mid 1A1 \mid \varepsilon, A \rightarrow 0S0$

No.

For which of the following sets of productions P is $(\{ S, A \}, \{ 0, 1 \}, P, S)$ an ambiguous grammar?

2. $S \rightarrow S \mid \varepsilon$

Yes: $S \Rightarrow_{\mathsf{lm}} \varepsilon$ and $S \Rightarrow_{\mathsf{lm}} S \Rightarrow_{\mathsf{lm}} \varepsilon$.

For which of the following sets of productions P is $(\{ S, A \}, \{ 0, 1 \}, P, S)$ an ambiguous grammar?

4. $S \to 1S1 \mid 1A1 \mid \varepsilon, A \to 1A1 \mid S$

Yes: $S \Rightarrow_{\mathsf{lm}} 1A1 \Rightarrow_{\mathsf{lm}} 1S1 \Rightarrow_{\mathsf{lm}} 11$ and $S \Rightarrow_{\mathsf{lm}} 1S1 \Rightarrow_{\mathsf{lm}} 11$.

Ambiguity

- It is common to interpret $7 - 3 - 2$ as $(7 - 3) - 2$.
- The minus operator is said to "associate to the left".
- Exponentiation typically associates to the right: $3^{3^3} = 3^{(3^3)}$.

Ambiguity

- It is also common to interpret $7 \cdot 3 - 2$ as $(7 \cdot 3) - 2$, and not $7 \cdot (3 - 2)$.
- The multiplication operator is said to "bind tighter than" the subtraction operator, or to have "higher precedence".

Ambiguity

The following (underspecified) context-free grammar over $\{+, -, \cdot, /, (,)\} \cup \{0, 1, \ldots, 9\}$ is unambiguous:

$$
\begin{aligned}
Expr &\rightarrow Term\ Add\text{-}op\ Expr \mid Term \\
Term &\rightarrow Term\ Mul\text{-}op\ Factor \mid Factor \\
Factor &\rightarrow Digit \mid (\ Expr\ ) \\
Add\text{-}op &\rightarrow + \mid - \\
Mul\text{-}op &\rightarrow \cdot \mid / \\
Digit &\rightarrow 0 \mid 1 \mid \ldots \mid 9
\end{aligned}
$$

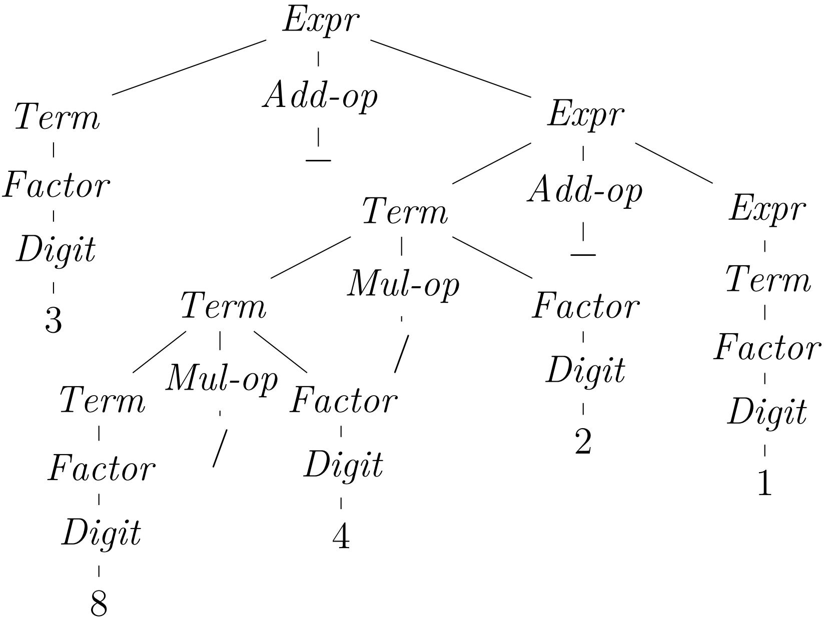
Use this grammar to parse the following string. Compute the value of the expression, using the parse tree to guide the evaluation.

$$3 - 8/4/2 - 1$$

$$\begin{array}{lcl} \mathit{Expr} & \to & \mathit{Term}\ \mathit{Add\text{-}op}\ \mathit{Expr} \mid \mathit{Term} \\ \mathit{Term} & \to & \mathit{Term}\ \mathit{Mul\text{-}op}\ \mathit{Factor} \mid \mathit{Factor} \\ \mathit{Factor} & \to & \mathit{Digit} \mid (\ \mathit{Expr}\ ) \\ \mathit{Add\text{-}op} & \to & + \mid - \\ \mathit{Mul\text{-}op} & \to & \cdot \mid / \\ \mathit{Digit} & \to & 0 \mid 1 \mid \ldots \mid 9 \end{array}$$

Respond at `https://pingo.coactum.de/729558`.

The parse tree



Right associative?

- Subtraction is right associative for this grammar: $3 - (((8/4)/2) - 1) = 3$.
- The usual way of parsing instead leads to $(3 - ((8/4)/2)) - 1 = 1$.
- One can make subtraction left associative by modifying the grammar:

$$Expr \to Expr\ Add\text{-}op\ Term \mid Term$$

Suggest some replacement for ??? that ensures that $3 \mathbin{\hat{}} 3 \mathbin{\hat{}} 3$ is a valid string that is interpreted as $3 \mathbin{\hat{}} (3 \mathbin{\hat{}} 3)$. The start symbol is E_0.

$$
\begin{array}{ll}
E_0 & \rightarrow E_0 \ \textit{Add-op} \ E_1 \mid E_1 \\
E_1 & \rightarrow E_1 \ \textit{Mul-op} \ E_2 \mid E_2 \\
E_2 & \rightarrow ??? \\
E_3 & \rightarrow \textit{Digit} \mid (\ E_0 \ ) \\
\textit{Add-op} & \rightarrow + \mid - \\
\textit{Mul-op} & \rightarrow \cdot \mid / \\
\textit{Digit} & \rightarrow 0 \mid 1 \mid \ldots \mid 9
\end{array}
$$

Respond at `https://pingo.coactum.de/729558`.

$$
\begin{aligned}
E_0 &\rightarrow E_0 \ \mathit{Add\text{-}op} \ E_1 \mid E_1 \\
E_1 &\rightarrow E_1 \ \mathit{Mul\text{-}op} \ E_2 \mid E_2 \\
E_2 &\rightarrow E_3 \ \hat{} \ E_2 \mid E_3 \\
E_3 &\rightarrow \mathit{Digit} \mid (\ E_0 \ ) \\
\mathit{Add\text{-}op} &\rightarrow + \mid - \\
\mathit{Mul\text{-}op} &\rightarrow \cdot \mid / \\
\mathit{Digit} &\rightarrow 0 \mid 1 \mid ... \mid 9
\end{aligned}
$$

Ambiguity

- It is undecidable whether a context-free grammar is ambiguous.
- However, several parser generators use restricted context-free grammars that are guaranteed to be unambiguous.
- If such a tool complains about a "conflict", then the problem might be that the grammar is ambiguous.

Ambiguity

- There are context-free languages for which there are no unambiguous context-free grammars.
- Such languages are called *inherently ambiguous*.
- See the book for an example.

Designing grammars

Define a grammar for some simple (context-free) language, perhaps a tiny programming language. Try to make the grammar unambiguous.

Designing grammars

If you want to know more about the use of grammars in the specification and implementation of programming languages you might be interested in the course *Programming language technology*.

Today

- Context-free languages.
- Some equivalences.
- Ambiguity.
- Designing grammars.

Next lecture

- Grammar transformations.
- Chomsky normal form.
- The pumping lemma for context-free languages.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-02-22

Today

- Grammar transformations.
- Chomsky normal form.
- The pumping lemma for context-free languages.

Grammar transformations

Grammar transformations

- A number of transformations of grammars.
- Will be used for parsing (next lecture).
- I have taken some information and terminology from "To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm" by Lange and Leiß.

$\textsc{Bin}$

- Result: No production $A \to \alpha$ where $|\alpha| \geq 3$.
- Replace each production $A \to X_1 X_2 ... X_n$, where $n \geq 3$, with:

$$\begin{aligned} A &\to X_1 A_2 \\ A_2 &\to X_2 A_3 \\ &\vdots \\ A_{n-1} &\to X_{n-1} X_n \end{aligned}$$

  Here $A_2, ..., A_{n-1}$ are new nonterminals.
- $L(\textsc{Bin}(G)) = L(G)$.

Del

- Result: No "deletion rules",
  i.e. productions of the form $A \to \varepsilon$.
- A nonterminal A is *nullable* if $A \Rightarrow^* \varepsilon$.

Del

Some examples:

- The production $A \to \varepsilon$ is removed.
- The production $A \to \alpha B \beta C \gamma$, where B and C are the only nullable nonterminals, is replaced with

$$A \to \alpha B \beta C \gamma,$$
$$A \to \alpha \beta C \gamma,$$
$$A \to \alpha B \beta \gamma \text{ and,}$$
$$\text{if } \alpha\beta\gamma \neq \varepsilon,\ A \to \alpha\beta\gamma.$$

$\textsc{Del}$

- The new productions are not deletion rules.
- If we do this for every production, then no nonterminal will be nullable, and $L(\textsc{Del}(G), A) = L(G, A) \setminus \{\, \varepsilon \,\}$.
- $L(\textsc{Del}(G)) = L(G) \setminus \{\, \varepsilon \,\}$.

DEL

Example:

- Before:

$$\begin{aligned} S &\to 0 \mid ABS \\ A &\to \varepsilon \mid BA \\ B &\to S \mid \varepsilon \end{aligned}$$

- After:

$$\begin{aligned} S &\to 0 \mid ABS \mid AS \mid BS \mid S \\ A &\to BA \mid B \mid A \\ B &\to S \end{aligned}$$

Del

If Del is applied to the following grammar, how many productions does the resulting grammar contain?

$$(\{\, S, A \,\}, \{\, 0 \,\}, (S \to (SA)^{10} \mid \varepsilon, A \to 0), S)$$

Respond at `https://pingo.coactum.de/729558`.

Del

- The Del transformation can make the grammar much larger.
- If every production $A \to \alpha$ satisfies $|\alpha| \leq 2$, then the blowup is contained.
- Run Bin before Del.

Unit

- Result: No unit productions (productions of the form $A \to B$).
- (A, B) is a *unit pair* if $A = B$ or $A \to C_1 \to \cdots \to C_n \to B$ (where $n \in \mathbb{N}$).
- Include exactly the following productions:

$$\begin{aligned}\{A \to \alpha \mid (A, B) &\text{ is a unit pair,}\\ &B \to \alpha \in P,\\ &\alpha \text{ is not a single nonterminal}\}\end{aligned}$$

UNIT

Example:

- Before:

$$\begin{aligned} A &\to 1 \mid B \\ B &\to 2 \mid C \\ C &\to AB \end{aligned}$$

- After:

$$\begin{aligned} A &\to 1 \mid 2 \mid AB \\ B &\to 2 \mid AB \\ C &\to AB \end{aligned}$$

$L(\textsc{Unit}(G)) = L(G)$.

Unit

The resulting grammar could be much larger than the original one:

$$
\begin{aligned}
A_1 &\to A_2 \mid 1 \\
A_2 &\to A_3 \mid 2 \\
A_3 &\to A_4 \mid 3 \\
&\vdots \\
A_n &\to A_1 \mid n
\end{aligned}
$$

Unit

The resulting grammar could be much larger than the original one:

$$
\begin{aligned}
A_1 &\to 1 \mid 2 \mid 3 \mid \ldots \mid n \\
A_2 &\to 1 \mid 2 \mid 3 \mid \ldots \mid n \\
A_3 &\to 1 \mid 2 \mid 3 \mid \ldots \mid n \\
&\vdots \\
A_n &\to 1 \mid 2 \mid 3 \mid \ldots \mid n
\end{aligned}
$$

Construct a grammar G for which $\textsc{Del}(\textsc{Unit}(G))$ contains a unit production.

Construct a grammar G for which $\textsc{Del}(\textsc{Unit}(G))$ contains a unit production.

Run $\textsc{Del}$ before $\textsc{Unit}$.

UNIT does not affect the following (underspecified) grammar:

$$S \to AB$$
$$A \to \varepsilon$$

If DEL is applied to it, then we get a grammar with a unit production:

$$S \to AB \mid B$$

Term

- Result: No terminals in productions $A \to \alpha$ where $|\alpha| \geq 2$.
- Find all terminals in such productions.
- For each such terminal b, add a new nonterminal B with a single production $B \to b$, and substitute B for b in every production $A \to \alpha$ where $|\alpha| \geq 2$.

$\textsc{Term}$

Example:

- Before:

$$
\begin{aligned}
A &\to A1 \mid 1 \mid 2 \\
B &\to 1
\end{aligned}
$$

- After:

$$
\begin{aligned}
A &\to AO \mid 1 \mid 2 \\
B &\to 1 \\
O &\to 1
\end{aligned}
$$

$L(\textsc{Term}(G)) = L(G)$.

$\textsc{Bin}/\textsc{Term}$

- I have written $\textsc{Bin}(G)$ and $\textsc{Term}(G)$, as if $\textsc{Bin}$ and $\textsc{Term}$ were functions.
- However, these transformations are not functions, because the names of the new nonterminals are not uniquely specified.
- Below I will pretend that the transformations are functions.

Chomsky normal form

Chomsky normal form

- A context-free grammar is in *Chomsky normal form* if every production is of the form $A \to BC$ or $A \to a$.
- For any context-free grammar G the grammar $G' = \textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$ is in Chomsky normal form and satisfies $L(G') = L(G) \setminus \{ \varepsilon \}$.

Chomsky normal form

- A context-free grammar is in *Chomsky normal form* if every production is of the form $A \to BC$ or $A \to a$.
- For any context-free grammar G the grammar $G' = \textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$ is in Chomsky normal form and satisfies $L(G') = L(G) \setminus \{ \varepsilon \}$.

I dropped the text book's requirement that there should be no useless symbols.

Consider the grammar $G = (\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A \mid S$$
$$A \to 1S \mid \varepsilon$$

- Is G ambiguous?
- Is $\textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$ ambiguous?

Respond at `https://pingo.coactum.de/729558`.

Consider the grammar
$G = (\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A \mid S$$
$$A \to 1S \mid \varepsilon$$

- Is G ambiguous?

Yes:

- $S \Rightarrow_{\mathsf{lm}} 0A \Rightarrow_{\mathsf{lm}} 0$.
- $S \Rightarrow_{\mathsf{lm}} S \Rightarrow_{\mathsf{lm}} 0A \Rightarrow_{\mathsf{lm}} 0$.

Consider the grammar
$G = (\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A \mid S$$
$$A \to 1S \mid \varepsilon$$

- Is $\textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$ ambiguous?

No:

- $\textsc{Bin}(G) = G$.

Consider the grammar $G = (\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A \mid S$$
$$A \to 1S \mid \varepsilon$$

- Is $\textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$ ambiguous?

No:

- $\textsc{Del}(\textsc{Bin}(G))$:

$$S \to 0A \mid 0 \mid S$$
$$A \to 1S$$

Consider the grammar $G = (\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A \mid S$$
$$A \to 1S \mid \varepsilon$$

- Is $\textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$ ambiguous?

No:

- $\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G)))$:

$$S \to 0A \mid 0$$
$$A \to 1S$$

Consider the grammar
$G = (\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A \mid S$$
$$A \to 1S \mid \varepsilon$$

- Is $\textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$ ambiguous?

No:

- $\textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$:

$$S \to ZA \mid 0 \qquad Z \to 0$$
$$A \to OS \qquad O \to 1$$

Consider the grammar
$G = (\{ S, A \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$S \to 0A \mid S$$
$$A \to 1S \mid \varepsilon$$

- Is $\textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$ ambiguous?

If G is ambiguous, then $\textsc{Unit}(G)$ is sometimes ambiguous, sometimes not.

The pumping lemma

The pumping lemma for CFLs

For every context-free language L over the alphabet Σ:

$$
\begin{aligned}
&\exists m \in \mathbb{N}.\\
&\quad \forall w \in L.\ |w| \geq m \Rightarrow\\
&\quad\quad \exists r, s, t, u, v \in \Sigma^*.\\
&\quad\quad\quad w = rstuv \wedge |stu| \leq m \wedge su \neq \varepsilon \wedge\\
&\quad\quad\quad \forall n \in \mathbb{N}.\ rs^n tu^n v \in L
\end{aligned}
$$

The pumping lemma for CFLs

For every context-free language L over the alphabet Σ:

$$
\begin{array}{l}
\exists m \in \mathbb{N}. \\
\quad \forall w \in L.\ |w| \geq m \Rightarrow \\
\quad\quad \exists r, s, t, u, v \in \Sigma^*. \\
\quad\quad\quad w = rstuv \wedge |stu| \leq m \wedge su \neq \varepsilon \wedge \\
\quad\quad\quad \forall n \in \mathbb{N}.\ rs^n tu^n v \in L
\end{array}
$$

Height

The height of a parse tree in $P(G, A)$ is the largest number of nonterminals encountered on any path from the root to a leaf.

$$height\left(\begin{array}{ccc} & A & \\ \diagup & | & \diagdown \\ A & A & 0 \\ | & | & \\ 0 & 0 & \end{array}\right) = 2$$

Height

For context-free grammars in
Chomsky normal form:

$$\forall p \in P(G, A).\ |yield(p)| \leq 2^{height(p)-1}$$

Height

For context-free grammars in Chomsky normal form:

$$\forall p \in P(G, A).\ |yield(p)| \leq 2^{height(p)-1}$$

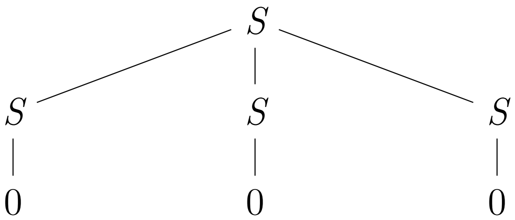
Proof: Exercise.

Height

Consider the following grammar and parse tree:

$$(\{\, S \,\}\,, \{\, 0 \,\}\,, (S \to SSS \mid 0), S)$$

$p =$



We have
$|yield(p)| = |000| = 3 \not\leq 2 = 2^{2-1} = 2^{height(p)-1}$.

The pumping lemma for CFLs

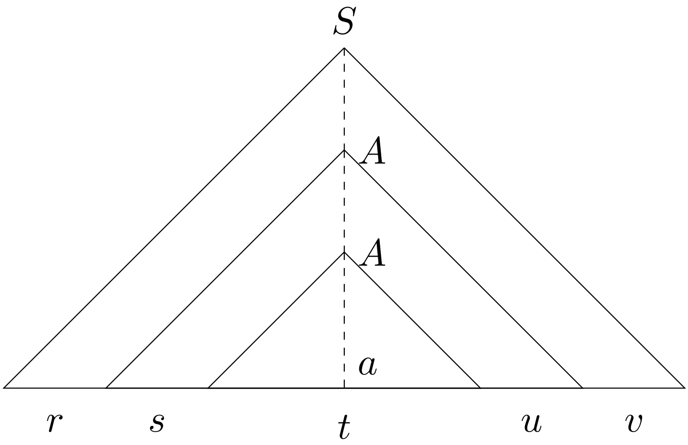
Proof sketch:

- Take any context-free grammar G for L.
- Let $G' = \textsc{Term}(\textsc{Unit}(\textsc{Del}(\textsc{Bin}(G))))$.
- If $G' = (N, \Sigma, P, S)$, let $m = 2^{|N|}$.
- Given a string $w \in L$ with $|w| \geq m$ we know that $w \neq \varepsilon$, so we have $w \in L \setminus \{ \varepsilon \} = L(G')$.

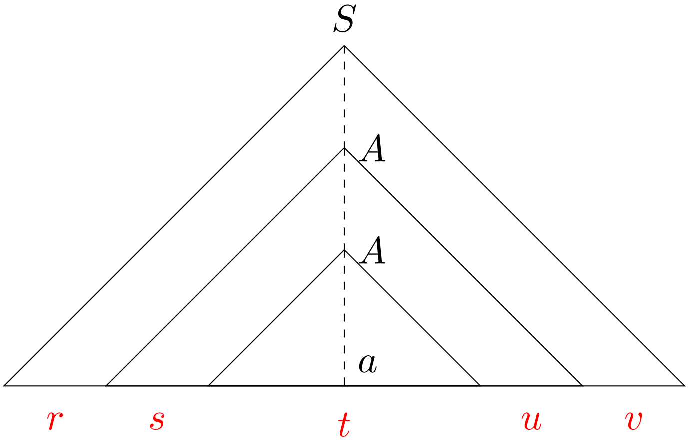
The pumping lemma for CFLs

- Take any parse tree p for w with respect to G'.
- We know that $2^{|N|} = m \leq |w| = |yield(p)| \leq 2^{height(p)-1}$, so $height(p) > |N|$.
- Take a path of maximal length (number of nonterminals) from the root of p to a leaf.
- Such a path must contain at least $|N| + 1$ nonterminals.
- By the pigeonhole principle the path must contain two instances of the same nonterminal, at most $|N| + 1$ steps from the leaf.

The pumping lemma for CFLs

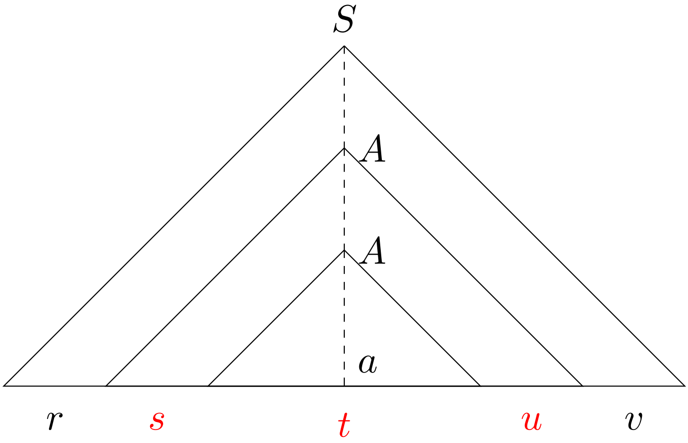


The pumping lemma for CFLs



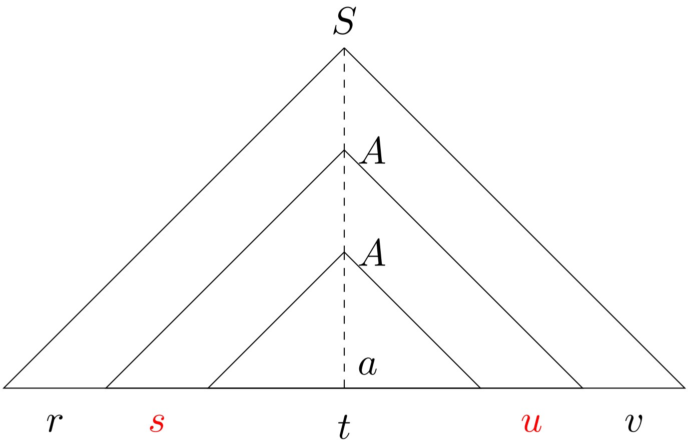
$w = rstuv$

The pumping lemma for CFLs



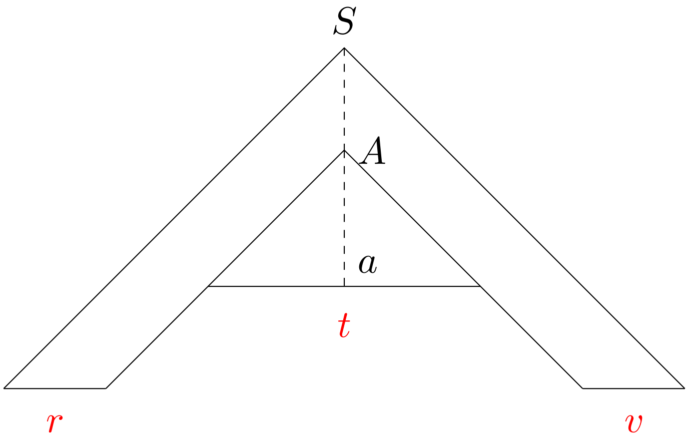
$$|stu| \leq 2^{(|N|+1)-1} = 2^{|N|} = m$$

The pumping lemma for CFLs



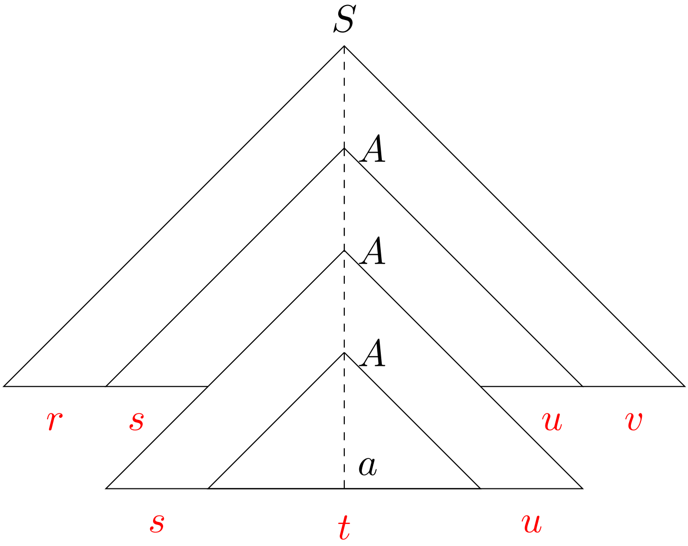
No nonterminal is nullable, $A \to BC \Rightarrow$
$s \neq \varepsilon \vee u \neq \varepsilon \Rightarrow su \neq \varepsilon$

The pumping lemma for CFLs



$rtv \in L(G') \subseteq L$

The pumping lemma for CFLs



$$rs^2tu^2v \in L(G') \subseteq L$$

The pumping lemma for CFLs

The language $L = \{ 0^n 1^n 2^n \mid n \in \mathbb{N} \}$ over $\Sigma = \{ 0, 1, 2 \}$ is not context-free. Proof sketch:

- Assume that L is context-free.
- Take the constant $m \in \mathbb{N}$ that we get from the pumping lemma.
- Consider the string $w = 0^m 1^m 2^m \in L$.
- Because $|w| \geq m$ we get some information:

$$\begin{aligned} &\exists r, s, t, u, v \in \{ 0, 1, 2 \}^* . \\ &\quad 0^m 1^m 2^m = rstuv \wedge |stu| \leq m \wedge \\ &\quad su \neq \varepsilon \wedge \forall n \in \mathbb{N}.\ rs^n tu^n v \in L \end{aligned}$$

The pumping lemma for CFLs

- Because $|w| \geq m$ we get some information:

$$\exists r, s, t, u, v \in \{\, 0, 1, 2 \,\}^* .$$
$$0^m 1^m 2^m = rstuv \wedge |stu| \leq m \wedge$$
$$su \neq \varepsilon \wedge \forall n \in \mathbb{N}.\ rs^n tu^n v \in L$$

- Because $|stu| \leq m$ this substring cannot contain both 0 and 2.
- Because $su \neq \varepsilon$ either s or u must contain at least one symbol from $\{\, 0, 1, 2 \,\}$.
- Thus rtv does not contain the same number of each symbol from $\{\, 0, 1, 2 \,\}$, so $rtv \notin L$.
- But $rtv = rs^0 tu^0 v \in L$, so we have found a contradiction.

Which of the following languages are context-free?

1. $\{0^n \mid n \in \mathbb{N}\}$.
2. $\{0^n 1^n \mid n \in \mathbb{N}\}$.
3. $\{0^n 1^n 2^n 3^n \mid n \in \mathbb{N}\}$.
4. $\{w \in \{\, 0, 1, 2 \,\}^* \mid \#_0(w) = \#_1(w) = \#_2(w)\}$.

Respond at `https://pingo.coactum.de/729558`.

Which of the following languages are context-free?

1. $\{0^n \mid n \in \mathbb{N}\}$.

Yes, this language is regular.

Which of the following languages are context-free?

2. $\{0^n 1^n \mid n \in \mathbb{N}\}$.

Yes, see a previous lecture.

Which of the following languages are context-free?

3. $\{0^n 1^n 2^n 3^n \mid n \in \mathbb{N}\}$.

No, use the pumping lemma with the string $0^m 1^m 2^m 3^m$.

Which of the following languages are context-free?

4. $\{w \in \{\,0, 1, 2\,\}^* \mid \#_0(w) = \#_1(w) = \#_2(w)\}$.

No, use the pumping lemma with the string $0^m1^m2^m$.

Today

- Grammar transformations.
- Chomsky normal form.
- The pumping lemma for context-free languages.

Next lecture

- Closure properties.
- Algorithms.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-02-26

Today

- Closure properties for context-free languages.
- Some algorithms for context-free languages.
- Some undecidable problems.

Closure properties

Context-free languages

- Every regular language is context-free.
- Exercise: Prove this.

Substitutions

Assume that

- Σ_1 and Σ_2 are alphabets and
- $F \in \Sigma_1 \to \wp(\Sigma_2^*)$.

The function F maps symbols to languages.
It can be lifted to words and languages:

$$\begin{aligned} F &\in \Sigma_1^* \to \wp(\Sigma_2^*) \\ F(\varepsilon) &= \{ \varepsilon \} \\ F(aw) &= F(a)F(w) \end{aligned}$$

$$\begin{aligned} F &\in \wp(\Sigma_1^*) \to \wp(\Sigma_2^*) \\ F(L) &= \bigcup_{w \in L} F(w) \end{aligned}$$

Substitutions

Example:

$$
\begin{aligned}
&F \in \{\, 0, 1 \,\} \to \wp(\{\, a, b \,\}^*) \\
&F(0) = \{\, a \,\} \\
&F(1) = \{\, a, b \,\}
\end{aligned}
$$

$$
\begin{aligned}
F(01) &= \{\, a \,\}\{\, a, b \,\}\{\, \varepsilon \,\} = \{\, aa, ab \,\} \\
F(\{\, 0, 01 \,\}) &= F(0) \cup F(01) = \{\, a \,\} \cup \{\, aa, ab \,\} \\
&= \{\, a, aa, ab \,\}
\end{aligned}
$$

What is $F(\{ 01 \}^*)$ when $F(0) = \{ a \}$ and $F(1) = \{ b, c \}$?

1. $\{ a, b, c \}^*$
2. $\{ abc \}^*$
3. $\{ ab, ac \}^*$
4. $\{ ac, bc \}^*$
5. $\{ a \}^* \{ b, c \}^*$
6. $\{ a, b \}^* \{ c \}^*$
7. $\{ a \}^* \{ bc \}^*$
8. $\{ ab \}^* \{ c \}^*$

Respond at `https://pingo.coactum.de/729558`.

$$
\begin{aligned}
F(\{\, 01 \,\}^*) &= \\
F(\textstyle\bigcup_{n\in\mathbb{N}} \{\, 01 \,\}^n) &= \\
F(\textstyle\bigcup_{n\in\mathbb{N}} \{\, (01)^n \,\}) &= \\
F(\{\, (01)^n \mid n \in \mathbb{N} \,\}) &= \\
\textstyle\bigcup_{w\in\{\, (01)^n \mid n\in\mathbb{N} \,\}} F(w) &= \\
\textstyle\bigcup_{n\in\mathbb{N}} F((01)^n) &= \\
\textstyle\bigcup_{n\in\mathbb{N}}(F(0)F(1))^n &= \\
(F(0)F(1))^* &= \\
(\{\, a \,\}\{\, b, c \,\})^* &= \\
\{\, ab, ac \,\}^*
\end{aligned}
$$

Closure under substitutions

If

- Σ_1 and Σ_2 are alphabets,
- $L \subseteq \Sigma_1^*$ is context-free,
- $F \in \Sigma_1 \to \wp(\Sigma_2^*)$, and
- $F(a)$ is context-free for every $a \in \Sigma_1$,

then $F(L)$ is context-free.

Closure under substitutions

Idea:

- Replace each terminal a in a grammar for L with the start symbol of a grammar for $F(a)$ (renaming nonterminals if necessary).

Example

Two alphabets and three context-free languages:

$$\Sigma_1 = \{\, 0, 1 \,\}$$
$$\Sigma_2 = \{\, a, b \,\}$$

$$\begin{aligned}
L \; &= L(G), \;\; G \; = (\{\, S \,\},\;\; \Sigma_1, S \to 0S1 \mid \varepsilon, S) \\
L_0 &= L(G_0), G_0 = (\{\, S_0 \,\}, \Sigma_2, S_0 \to ab, \qquad S_0) \\
L_1 &= L(G_1), G_1 = (\{\, S_1 \,\}, \Sigma_2, S_1 \to ba, \qquad S_1)
\end{aligned}$$

Example

- A function from Σ_1 to $\wp(\Sigma_2^*)$:

$$\begin{aligned}
&F \in \Sigma_1 \to \wp(\Sigma_2^*) \\
&F(0) = L_0 \\
&F(1) = L_1
\end{aligned}$$

- $F(L) = L(G')$, where $G' = (\{\, S, S_0, S_1 \,\}, \Sigma_2, P, S)$ and P contains the following productions:

$$\begin{aligned}
&S \to S_0 S S_1 \mid \varepsilon \\
&S_0 \to ab \\
&S_1 \to ba
\end{aligned}$$

Application

Let us prove that $L = \{ 0^n 1^n 2^n 3^n \mid n \in \mathbb{N} \}$ is not context-free.

- Assume that L is context-free.
- Then $F(L)$ is context-free:

$$
\begin{aligned}
&F \in \{ 0, 1, 2, 3 \} \to \wp(\{ 0, 1, 2 \}^*) \\
&F(0) = \{ 0 \} \\
&F(1) = \{ 1 \} \\
&F(2) = \{ 2 \} \\
&F(3) = \{ \varepsilon \}
\end{aligned}
$$

- But $F(L) = \{ 0^n 1^n 2^n \mid n \in \mathbb{N} \}$, which we know is not context-free.
- Thus L is not context-free.

Closure under union

- If L_1 and L_2 are context-free,
  then $L_1 \cup L_2$ is context-free.
- Substitute L_i for i in $\{ 1, 2 \}$.

Closure under union

Recall: $F(L)$ is context-free if

- Σ_1 and Σ_2 are alphabets,
- $L \subseteq \Sigma_1^*$ is context-free,
- $F \in \Sigma_1 \to \wp(\Sigma_2^*)$, and
- $F(a)$ is context-free for every $a \in \Sigma_1$.

In this case:

- $\Sigma_1 = \{ 1, 2 \}$, Σ_2 is the union of the sets of terminals of some grammars for L_1 and L_2.
- $L = \{ 1, 2 \} \subseteq \Sigma_1^*$ is context-free.
- $F(1) = L_1$, $F(2) = L_2$.
- $F(1)$ and $F(2)$ are context-free.

Thus $F(L) = L_1 \cup L_2$ is context-free.

Closure under concatenation

- If L_1 and L_2 are context-free, then $L_1 L_2$ is context-free.
- Substitute L_i for i in $\{ 12 \}$.

Closure under Kleene star

- If L is context-free,
  then L^* is context-free.
- Substitute L for 1 in $\{ 1 \}^*$.

Closure under Kleene plus

- If L is context-free,
  then L^+ is context-free.
- Substitute L for 1 in $\{\,1\,\}^+$.

Homomorphisms

Assume that

- Σ_1 and Σ_2 are alphabets and
- $h \in \Sigma_1 \to \Sigma_2^*$.

The function h maps symbols to words.
It can be lifted to words and languages:

$$\begin{aligned} & h \in \Sigma_1^* \to \Sigma_2^* \\ & h(\varepsilon) \quad = \varepsilon \\ & h(aw) = h(a)h(w) \end{aligned}$$

$$\begin{aligned} & h \in \wp(\Sigma_1^*) \to \wp(\Sigma_2^*) \\ & h(L) = \{\, h(w) \mid w \in L \,\} \end{aligned}$$

The function $h \in \Sigma_1^* \to \Sigma_2^*$ is a
string homomorphism.

Closure under homomorphism

- If $L \subseteq \Sigma_1^*$ is context-free, then $h(L)$ is context-free.
- Apply the substitution $F(a) = \{ h(a) \}$ to L.

Prove that $L = \{ \, 01^n23^n45^n6 \mid n \in \mathbb{N} \, \}$ is not a context-free language over $\{ \, 0, 1, 2, 3, 4, 5, 6 \, \}$.

You may use the fact that $\{ \, 0^n1^n2^n \mid n \in \mathbb{N} \, \}$ is not a context-free language over $\{ \, 0, 1, 2 \, \}$.

Hint: Can you find a string homomorphism h for which $h(L) = \{ \, 0^n1^n2^n \mid n \in \mathbb{N} \, \}$?

Prove that $L = \{ 01^n23^n45^n6 \mid n \in \mathbb{N} \}$ is not a context-free language over $\{ 0, 1, 2, 3, 4, 5, 6 \}$.

You may use the fact that $\{ 0^n1^n2^n \mid n \in \mathbb{N} \}$ is not a context-free language over $\{ 0, 1, 2 \}$.

Use the following homomorphism:

$$\begin{aligned} h(0) &= \varepsilon & \qquad h(4) &= \varepsilon \\ h(1) &= 0 & h(5) &= 2 \\ h(2) &= \varepsilon & h(6) &= \varepsilon \\ h(3) &= 1 \end{aligned}$$

Closure under intersection

- If L_1 and L_2 are context-free,
  then $L_1 \cap L_2$ is *not* necessarily context-free.
  - A counterexample:
    $L_1 = \{ 0^n 1^n \mid n \in \mathbb{N} \} \{ 2 \}^*$,
    $L_2 = \{ 0 \}^* \{ 1^n 2^n \mid n \in \mathbb{N} \}$.
- If L is a context-free language over Σ,
  then $\overline{L} = \Sigma^* \setminus L$ is *not* necessarily context-free.
  - Note that $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.
- If L_1 and L_2 are context-free,
  then $L_1 \setminus L_2$ is *not* necessarily context-free.

Closure under intersection

- If L is context-free and R is regular, then $L \cap R$ is context-free.
- If L is context-free and R is regular, then $L \setminus R$ is context-free.
  - Note that $L \setminus R = L \cap \overline{R}$.

If Σ is an alphabet, $R \subseteq \Sigma^*$ is regular and $L \subseteq \Sigma^*$ is context-free, what can we say about $R \setminus L$?

1. It is always regular.
2. It is not necessarily regular,
   but always context-free.
3. It is not necessarily context-free.

Hint: $\Sigma^* \setminus L = \overline{L}$.

Respond at `https://pingo.coactum.de/729558`.

If Σ is an alphabet, $R \subseteq \Sigma^*$ is regular and $L \subseteq \Sigma^*$ is context-free, what can we say about $R \setminus L$?

1. It is always regular.
2. It is not necessarily regular, but always context-free.
3. It is not necessarily context-free.

3: If $R = \Sigma^*$, then $R \setminus L = \Sigma^* \setminus L = \overline{L}$, and $\overline{L}$ is not necessarily context-free.

Some algorithms

Testing emptiness

For any context-free language L,
given as a context-free grammar $G = (N, \Sigma, P, S)$,
we can decide if $L = \emptyset$:

- A symbol $X \in N \cup \Sigma$ is *generating* if $X \Rightarrow^* w$ for some $w \in \Sigma^*$.
- $L = \emptyset$ if and only if S is not generating.

Computing the generating symbols

The set of generating symbols can be computed (perhaps inefficiently) in the following way:

- Let the function $step \in \wp(N \cup \Sigma) \to \wp(N \cup \Sigma)$ be defined by

$$step(\Gamma) = \left\{ A \;\middle|\; \begin{array}{l} A \to \alpha \in P, \\ \text{every symbol in } \alpha \text{ is in } \Gamma \end{array} \right\}.$$

- Initialise Γ to Σ.
- Repeat until $step(\Gamma) \subseteq \Gamma$:
  - Set Γ to $\Gamma \cup step(\Gamma)$.
- Return Γ.

Computing the generating symbols

$$(\{ S, A, B \}, \{ 0, 1 \}, \{ S \to S, A \to B, B \to 0 \}, S)$$

- Initialisation: $\Gamma_0 = \{ 0, 1 \}$.
- Step 1: $\mathit{step}(\Gamma_0) = \{ B \}$, $\Gamma_1 = \{ 0, 1, B \}$.
- Step 2:
  $\mathit{step}(\Gamma_1) = \{ A, B \}$, $\Gamma_2 = \{ 0, 1, A, B \}$.
- Done: $\mathit{step}(\Gamma_2) = \{ A, B \} \subseteq \Gamma_2$.

Compute the generating symbols of the grammar $(\{ S, A, B \}, \{ 0, 1 \}, P, S)$, where P is defined in the following way:

$$
\begin{aligned}
S &\to 0A \mid B \\
A &\to 1S \mid \varepsilon \\
B &\to AB
\end{aligned}
$$

1. S.
2. A.
3. B.
4. 0.
5. 1.

Respond at `https://pingo.coactum.de/729558`.

$$
\begin{aligned}
S &\to 0A \mid B \\
A &\to 1S \mid \varepsilon \\
B &\to AB
\end{aligned}
$$

1. $\Gamma = \{\, 0, 1 \,\}$.
2. $\Gamma = \{\, 0, 1 \,\} \cup \{\, A \,\}$.
3. $\Gamma = \{\, 0, 1, A \,\} \cup \{\, S, A \,\}$.

Testing if the empty string is a member

For any context-free language L,
given as a context-free grammar $G = (N, \Sigma, P, S)$,
we can decide if $\varepsilon \in L$:

- A nonterminal $A \in N$ is *nullable* if $A \Rightarrow^* \varepsilon$.
- We have $\varepsilon \in L$ if and only if S is nullable.

Computing the nullable nonterminals

The set of nullable nonterminals can be computed (perhaps inefficiently) in the following way:

- Let the function $step \in \wp(N) \to \wp(N)$ be defined by

$$step(E) = \left\{ A \,\middle|\, \begin{array}{l} A \to \alpha \in P, \\ \text{every symbol in } \alpha \text{ is a} \\ \text{nonterminal in } E \end{array} \right\}.$$

- Initialise E to $\emptyset$.
- Repeat until $step(E) \subseteq E$:
  - Set E to $E \cup step(E)$.
- Return E.

Computing the nullable nonterminals

$$(\{ S, A, B \}, \{ 0, 1 \}, \{ S \to AA, A \to \varepsilon \}, S)$$

- Initialisation: $\Gamma_0 = \emptyset$.
- Step 1: $step(\Gamma_0) = \{ A \}$, $\Gamma_1 = \{ A \}$.
- Step 2: $step(\Gamma_1) = \{ S, A \}$, $\Gamma_2 = \{ S, A \}$.
- Done: $step(\Gamma_2) = \{ S, A \} \subseteq \Gamma_2$.

The CYK algorithm

For any context-free language L,
given as a context-free grammar $G = (\_, \Sigma, \_, \_)$,
and for any nonempty string $w \in \Sigma^*$,
we can decide if $w \in L$.

The CYK algorithm

- Convert G to a grammar $G' = (N, \Sigma, P, S)$ in Chomsky normal form such that $w \in L(G') \Leftrightarrow w \in L(G)$.
- Build a CYK table T for G' and w:
  - $T_{i,j}$ is defined for $i, j \in \{1, ..., |w|\}$ satisfying $i \leq j$.
  - $T_{i,j} = \left\{ A \in N \;\middle|\; A \Rightarrow^* w_i ... w_j \right\}$, where w_i denotes the i-th symbol in w (counting from 1).
- Check if $S \in T_{1,|w|}$.

The CYK algorithm

The table can be computed in the following way:

- First set

  $$T_{i,i} = \{\, A \mid A \to w_i \in P \,\}$$

  for each $i \in \{1, \ldots, |w|\}$.
- Then set

  $$T_{i,j} = \left\{ A \;\middle|\; \begin{array}{l} k \in \{\, i, \ldots, j-1 \,\}\,, \\ B \in T_{i,k}, C \in T_{k+1,j}, \\ A \to BC \in P \end{array} \right\}$$

  for all $i, j \in \{1, \ldots, |w|\}$ satisfying $j - i + 1 = 2$.
- Repeat the previous step for $j - i + 1 = 3$, 4 and so on up to $|w|$.

The CYK algorithm

An example of dynamic programming.

The CYK algorithm

$$
\begin{aligned}
&S \to AA \\
&A \to AB \mid 0 \\
&B \to 1 \mid 2
\end{aligned}
$$

0	1	2

The CYK algorithm

$$
\begin{aligned}
S &\to AA \\
A &\to AB \mid 0 \\
B &\to 1 \mid 2
\end{aligned}
$$

$\{A\}$		
0	1	2

The CYK algorithm

$$S \to AA$$
$$A \to AB \mid 0$$
$$B \to 1 \mid 2$$

$$\begin{array}{ccc} \{\,A\,\} & \{\,B\,\} & \\ \hline 0 & 1 & 2 \end{array}$$

The CYK algorithm

$$S \to AA$$
$$A \to AB \mid 0$$
$$B \to 1 \mid 2$$

$$\begin{array}{ccc} \{\,A\,\} & \{\,B\,\} & \{\,B\,\} \\ \hline 0 & 1 & 2 \end{array}$$

The CYK algorithm

$$
\begin{aligned}
S &\to AA \\
A &\to AB \mid 0 \\
B &\to 1 \mid 2
\end{aligned}
$$

$$
\begin{array}{ccc}
\{\,A\,\} & & \\
\{\,A\,\} & \{\,B\,\} & \{\,B\,\} \\
\hline
0 & 1 & 2
\end{array}
$$

The CYK algorithm

$$S \to AA$$
$$A \to AB \mid 0$$
$$B \to 1 \mid 2$$

$$\begin{array}{ccc}
\{\,A\,\} & \emptyset & \\
\{\,A\,\} & \{\,B\,\} & \{\,B\,\} \\
\hline
0 & 1 & 2
\end{array}$$

The CYK algorithm

$$
\begin{aligned}
S &\to AA \\
A &\to AB \mid 0 \\
B &\to 1 \mid 2
\end{aligned}
$$

$$
\begin{array}{ccc}
\{\, A \,\} & & \\
\{\, A \,\} & \emptyset & \\
\{\, A \,\} & \{\, B \,\} & \{\, B \,\} \\
\hline
0 & 1 & 2
\end{array}
$$

Consider the grammar $(\{ S, T, U, Z, O \}, \{ 0, 1 \}, P, S)$, where P contains the following productions:

$$
\begin{aligned}
S &\to ZT \mid OU \qquad & Z &\to 0 \\
T &\to SZ \mid 0 \qquad & O &\to 1 \\
U &\to SO \mid 1
\end{aligned}
$$

Note that the grammar is in Chomsky normal form.

1. Construct a CYK table for the string 0110.
2. Construct a parse tree with S in the root and yield 0110.

$$
\begin{array}{ll}
S \to ZT \mid OU & \qquad Z \to 0 \\
T \to SZ \mid 0 & \qquad O \to 1 \\
U \to SO \mid 1 &
\end{array}
$$

0	1	1	0

$$
\begin{array}{ll}
S \to ZT \mid OU & \qquad Z \to 0 \\
T \to SZ \mid 0 & \qquad O \to 1 \\
U \to SO \mid 1 &
\end{array}
$$

$$
\begin{array}{cccc}
\{\, T, Z \,\} & & & \\
\hline
0 & 1 & 1 & 0
\end{array}
$$

$$
\begin{aligned}
S &\to ZT \mid OU & \qquad Z &\to 0 \\
T &\to SZ \mid 0 & \qquad O &\to 1 \\
U &\to SO \mid 1 & &
\end{aligned}
$$

$\{T, Z\}$	$\{U, O\}$		
0	1	1	0

$$
\begin{array}{lll}
S \to ZT \mid OU & \qquad & Z \to 0 \\
T \to SZ \mid 0 & & O \to 1 \\
U \to SO \mid 1 & &
\end{array}
$$

$$
\begin{array}{cccc}
\{\, T, Z \,\} & \{\, U, O \,\} & \{\, U, O \,\} & \\
\hline
0 & 1 & 1 & 0
\end{array}
$$

$$S \to ZT \mid OU \qquad Z \to 0$$
$$T \to SZ \mid 0 \qquad O \to 1$$
$$U \to SO \mid 1$$

$$\frac{\{\,T, Z\,\} \quad \{\,U, O\,\} \quad \{\,U, O\,\} \quad \{\,T, Z\,\}}{0 \qquad\qquad 1 \qquad\qquad 1 \qquad\qquad 0}$$

$$
\begin{aligned}
S &\to ZT \mid OU \qquad & Z &\to 0 \\
T &\to SZ \mid 0 & O &\to 1 \\
U &\to SO \mid 1
\end{aligned}
$$

$$
\begin{array}{cccc}
\emptyset & & & \\
\{T, Z\} & \{U, O\} & \{U, O\} & \{T, Z\} \\
\hline
0 & 1 & 1 & 0
\end{array}
$$

$$
\begin{array}{ll}
S \to ZT \mid OU & \qquad Z \to 0 \\
T \to SZ \mid 0 & \qquad O \to 1 \\
U \to SO \mid 1 &
\end{array}
$$

$$
\begin{array}{cccc}
\emptyset & \{\, S \,\} & & \\
\{\, T, Z \,\} & \{\, U, O \,\} & \{\, U, O \,\} & \{\, T, Z \,\} \\
\hline
0 & 1 & 1 & 0
\end{array}
$$

$$
\begin{array}{ll}
S \to ZT \mid OU & \qquad Z \to 0 \\
T \to SZ \mid 0 & \qquad O \to 1 \\
U \to SO \mid 1 &
\end{array}
$$

$$
\begin{array}{cccc}
\emptyset & \{\, S \,\} & \emptyset & \\
\{\, T, Z \,\} & \{\, U, O \,\} & \{\, U, O \,\} & \{\, T, Z \,\} \\
\hline
0 & 1 & 1 & 0
\end{array}
$$

$$
\begin{aligned}
S &\to ZT \mid OU & \qquad Z &\to 0 \\
T &\to SZ \mid 0 & \qquad O &\to 1 \\
U &\to SO \mid 1 & &
\end{aligned}
$$

$$
\begin{array}{cccc}
\emptyset & & & \\
\emptyset & \{\, S \,\} & \emptyset & \\
\{\, T, Z \,\} & \{\, U, O \,\} & \{\, U, O \,\} & \{\, T, Z \,\} \\
\hline
0 & 1 & 1 & 0
\end{array}
$$

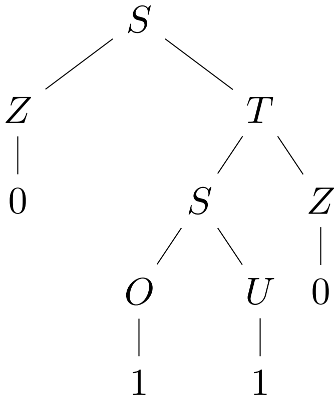
$$
\begin{array}{ll}
S \to ZT \mid OU & \qquad Z \to 0 \\
T \to SZ \mid 0 & \qquad O \to 1 \\
U \to SO \mid 1 &
\end{array}
$$

$$
\begin{array}{cccc}
\emptyset & \{\, T \,\} & & \\
\emptyset & \{\, S \,\} & \emptyset & \\
\{\, T, Z \,\} & \{\, U, O \,\} & \{\, U, O \,\} & \{\, T, Z \,\} \\
\hline
0 & 1 & 1 & 0
\end{array}
$$

$$
\begin{array}{ll}
S \to ZT \mid OU & \qquad Z \to 0 \\
T \to SZ \mid 0 & \qquad O \to 1 \\
U \to SO \mid 1 &
\end{array}
$$

$$
\begin{array}{cccc}
\{\,S\,\} & & & \\
\emptyset & \{\,T\,\} & & \\
\emptyset & \{\,S\,\} & \emptyset & \\
\{\,T, Z\,\} & \{\,U, O\,\} & \{\,U, O\,\} & \{\,T, Z\,\} \\
\hline
0 & 1 & 1 & 0
\end{array}
$$

$$
\begin{array}{cccc}
\{\,S\,\} & & & \\
\emptyset & \{\,T\,\} & & \\
\emptyset & \{\,S\,\} & \emptyset & \\
\{\,T,Z\,\} & \{\,U,O\,\} & \{\,U,O\,\} & \{\,T,Z\,\} \\
\hline
0 & 1 & 1 & 0
\end{array}
$$



The CYK algorithm

- A potential problem:
  The size of G' can be quadratic
  in the size of G.
- A variant of the algorithm that does not use the $\textsc{Unit}$ transformation can be devised:
  - Time complexity: $O(|G||w|^3)$.
  - Space complexity: $O(|G||w|^2)$.

  See Lange and Leiß.

Some undecidable problems

Some undecidable problems

The following things cannot, in general, be determined (using, say, a Turing machine):

- If a context-free grammar is ambiguous.
- If a context-free language, given by a context-free grammar, is *inherently* ambiguous.
- If $L(G_1) = L(G_2)$ for two context-free grammars G_1 and G_2.
- ...

Some undecidable problems

If you want to know more about why certain problems are undecidable, then you might be interested in the course *Computability*.

Today

- Closure properties for context-free languages.
- Some algorithms for context-free languages.
- Some undecidable problems.

Next lecture

- Pushdown automata.
- Turing machines.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-02-29

Today

- Pushdown automata.
- Turing machines.

Pushdown automata

Pushdown automata

- The class of regular languages can be defined using regular expressions or different kinds of automata.
- Is there a class of automata that defines the context-free languages?

Pushdown automata

A pushdown automaton (PDA) can be given as a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$:

- A finite set of states (Q).
- An alphabet (Σ with $\varepsilon \notin \Sigma$).
- A stack alphabet (Γ).
- A transition function ($\delta \in Q \times (\{\, \varepsilon \,\} \cup \Sigma^1) \times \Gamma \to \mathrm{Fin}(Q \times \Gamma^*)$).
- A start state ($q_0 \in Q$).
- A start symbol ($Z_0 \in \Gamma$).
- A set of accepting states ($F \subseteq Q$).

Pushdown automata

An *instantaneous description* (ID) for a given PDA is a triple (q, w, γ):

- The current state ($q \in Q$).
- The remainder of the input string ($w \in \Sigma^*$).
- The current stack ($\gamma \in \Gamma^*$).

Pushdown automata

The following relation between IDs defines what kinds of transitions are possible:

$$\frac{u \in \{\, \varepsilon \,\} \cup \Sigma^1 \quad (q, \alpha) \in \delta(p, u, Z)}{(p, uv, Z\gamma) \vdash (q, v, \alpha\gamma)}$$

The reflexive transitive closure of $\vdash$ can be defined inductively:

$$\frac{}{I \vdash^* I} \qquad\qquad \frac{I \vdash J \quad J \vdash^* K}{I \vdash^* K}$$

Pushdown automata

- A PDA:

$$(\{q\}, \{0\}, \{A, B\}, \delta, q, A, \emptyset)$$

$$\begin{array}{ll} \delta(q, \varepsilon, A) = \{(q, \varepsilon)\} & \delta(q, \varepsilon, B) = \{(q, \varepsilon)\} \\ \delta(q, 0, A) = \emptyset & \delta(q, 0, B) = \{(q, B)\} \end{array}$$

- Some possible transitions:

$$\begin{array}{l} (q, 00, A) \vdash (q, 00, \varepsilon) \\ (q, 00, B) \vdash (q, 0, B) \vdash (q, \varepsilon, B) \vdash (q, \varepsilon, \varepsilon) \\ (q, 00, B) \vdash (q, 0, B) \vdash (q, 0, \varepsilon) \\ (q, 00, B) \vdash (q, 00, \varepsilon) \end{array}$$

Consider the PDA $P = (\{q\}, \{0, 1\}, \{A, B\}, \delta, q, B, \{q\})$, where δ is defined in the following way:

$$\begin{array}{ll} \delta(q, \varepsilon, A) = \{(q, \varepsilon)\} & \delta(q, \varepsilon, B) = \{(q, BA)\} \\ \delta(q, 0, A) = \emptyset & \delta(q, 0, B) = \{(q, \varepsilon)\} \\ \delta(q, 1, A) = \emptyset & \delta(q, 1, B) = \{(q, AB)\} \end{array}$$

Which of the following propositions are true for P?

1. $(q, 01, AB) \vdash^* (q, \varepsilon, \varepsilon)$
2. $(q, 01, AB) \vdash^* (q, \varepsilon, AAA)$
3. $(q, 01, AB) \vdash^* (q, 1, \varepsilon)$
4. $(q, 01, AB) \vdash^* (q, 1, AAA)$

Respond at `https://pingo.coactum.de/729558`.

Consider the PDA $P = (\{q\}, \{0, 1\}, \{A, B\}, \delta, q, B, \{q\})$, where δ is defined in the following way:

$$\begin{array}{ll} \delta(q, \varepsilon, A) = \{(q, \varepsilon)\} & \delta(q, \varepsilon, B) = \{(q, BA)\} \\ \delta(q, 0, A) = \emptyset & \delta(q, 0, B) = \{(q, \varepsilon)\} \\ \delta(q, 1, A) = \emptyset & \delta(q, 1, B) = \{(q, AB)\} \end{array}$$

All possible transitions:

$$(q, 01, AB) \vdash (q, 01, B) \vdash^n (q, 01, BA^n) \vdash$$
$$(q, 1, A^n) \vdash^n (q, 1, \varepsilon)$$

Pushdown automata

The language of a PDA:

$$L((Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)) =$$
$$\{\, w \in \Sigma^* \mid q \in F, \gamma \in \Gamma^*, (q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma) \,\}$$

Consider the PDA $P = (\{q\}, \{0, 1\}, \{A, B\}, \delta, q, B, \{q\})$ again, where δ is still defined in the following way:

$$\begin{aligned} \delta(q, \varepsilon, A) &= \{(q, \varepsilon)\} & \delta(q, \varepsilon, B) &= \{(q, BA)\} \\ \delta(q, 0, A) &= \emptyset & \delta(q, 0, B) &= \{(q, \varepsilon)\} \\ \delta(q, 1, A) &= \emptyset & \delta(q, 1, B) &= \{(q, AB)\} \end{aligned}$$

Which of the following strings are members of $L(P)$?

1. 00
2. 01
3. 10
4. 11

Respond at `https://pingo.coactum.de/729558`.

Consider the PDA $P = (\{q\}, \{0,1\}, \{A,B\}, \delta, q, B, \{q\})$ again, where δ is still defined in the following way:

$$\begin{aligned} \delta(q,\varepsilon,A) &= \{(q,\varepsilon)\} & \delta(q,\varepsilon,B) &= \{(q,BA)\} \\ \delta(q,0,A) &= \emptyset & \delta(q,0,B) &= \{(q,\varepsilon)\} \\ \delta(q,1,A) &= \emptyset & \delta(q,1,B) &= \{(q,AB)\} \end{aligned}$$

Which of the following strings are members of $L(P)$?

1. 00

No. All possible transitions:

$$(q,00,B) \vdash^n (q,00,BA^n) \vdash (q,0,A^n) \vdash^n (q,0,\varepsilon)$$

Consider the PDA $P = (\{q\}, \{0, 1\}, \{A, B\}, \delta, q, B, \{q\})$ again, where δ is still defined in the following way:

$$\begin{aligned} \delta(q, \varepsilon, A) &= \{(q, \varepsilon)\} & \delta(q, \varepsilon, B) &= \{(q, BA)\} \\ \delta(q, 0, A) &= \emptyset & \delta(q, 0, B) &= \{(q, \varepsilon)\} \\ \delta(q, 1, A) &= \emptyset & \delta(q, 1, B) &= \{(q, AB)\} \end{aligned}$$

Which of the following strings are members of $L(P)$?

2. 01

No. All possible transitions:

$$(q, 01, B) \vdash^n (q, 01, BA^n) \vdash (q, 1, A^n) \vdash^n (q, 1, \varepsilon)$$

Consider the PDA $P = (\{q\}, \{0,1\}, \{A,B\}, \delta, q, B, \{q\})$ again, where δ is still defined in the following way:

$$\begin{array}{ll} \delta(q,\varepsilon,A) = \{(q,\varepsilon)\} & \delta(q,\varepsilon,B) = \{(q,BA)\} \\ \delta(q,0,A) = \emptyset & \delta(q,0,B) = \{(q,\varepsilon)\} \\ \delta(q,1,A) = \emptyset & \delta(q,1,B) = \{(q,AB)\} \end{array}$$

Which of the following strings are members of $L(P)$?

3. 10

Yes:

$$(q,10,B) \vdash (q,0,AB) \vdash (q,0,B) \vdash (q,\varepsilon,\varepsilon)$$

Consider the PDA $P = (\{q\}, \{0, 1\}, \{A, B\}, \delta, q, B, \{q\})$ again, where δ is still defined in the following way:

$$\begin{array}{ll} \delta(q, \varepsilon, A) = \{(q, \varepsilon)\} & \delta(q, \varepsilon, B) = \{(q, BA)\} \\ \delta(q, 0, A) = \emptyset & \delta(q, 0, B) = \{(q, \varepsilon)\} \\ \delta(q, 1, A) = \emptyset & \delta(q, 1, B) = \{(q, AB)\} \end{array}$$

Which of the following strings are members of $L(P)$?

4. 11

Yes:

$$(q, 11, B) \vdash (q, 1, AB) \vdash (q, 1, B) \vdash (q, \varepsilon, AB)$$

Pushdown automata

Another way to define a language for a PDA:

$$N((Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)) = \\ \{\, w \in \Sigma^* \mid q \in Q, (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \,\}$$

The following property holds for every language L:

$(\exists$ a PDA $P_1.\ L(P_1) = L) \Leftrightarrow$
$(\exists$ a PDA $P_2.\ N(P_2) = L)$

Grammars and automata

For any alphabet Σ (with $\varepsilon \notin \Sigma$) and language $L \subseteq \Sigma^*$ one can prove that the following two statements are equivalent:

- There is a context-free grammar G, with Σ as its set of terminals, satisfying $L(G) = L$.
- There is a pushdown automaton P with alphabet Σ satisfying $L(P) = L$.

Grammars and automata

Given a context-free grammar $G = (N, \Sigma, P, S)$ we can construct the PDA $Q = (\{\, q \,\}, \Sigma, N \cup \Sigma, \delta, q, S, \{\, q \,\})$, where δ is defined in the following way:

$$
\begin{aligned}
\delta(q, \varepsilon, A) &= \{\, (q, \alpha) \mid A \to \alpha \in P \,\} \\
\delta(q, a, a) &= \{\, (q, \varepsilon) \,\} \\
\delta(q, \_, \_) &= \emptyset
\end{aligned}
$$

Which of the following statements is true for $G = (\{ S \}, \{ 0, 1 \}, (S \to 0S \mid 1), S)$?

1. $L(G) = L(Q)$.
2. $L(G) = N(Q)$.

Respond at `https://pingo.coactum.de/729558`.

Which of the following statements is true for $G = (\{ S \}, \{ 0, 1 \}, (S \to 0S \mid 1), S)$?

1. $L(G) = L(Q)$.
2. $L(G) = N(Q)$.

$L(G) = L(0^*1)$.

Which of the following statements is true for $G = (\{ S \}, \{ 0, 1 \}, (S \to 0S \mid 1), S)$?

1. $L(G) = L(Q)$.
2. $L(G) = N(Q)$.

$$
\begin{aligned}
\delta(q, \varepsilon, S) &= \{ (q, 0S), (q, 1) \} \\
\delta(q, 0, 0) &= \{ (q, \varepsilon) \} \\
\delta(q, 1, 1) &= \{ (q, \varepsilon) \} \\
\delta(q, \_, \_) &= \emptyset
\end{aligned}
$$

Which of the following statements is true for $G = (\{ S \}, \{ 0, 1 \}, (S \to 0S \mid 1), S)$?

1. $L(G) = L(Q)$.
2. $L(G) = N(Q)$.

$$
\begin{aligned}
\delta(q, \varepsilon, S) &= \{ (q, 0S), (q, 1) \} \\
\delta(q, 0, 0) &= \{ (q, \varepsilon) \} \\
\delta(q, 1, 1) &= \{ (q, \varepsilon) \} \\
\delta(q, \_, \_) &= \emptyset
\end{aligned}
$$

1. No: $\varepsilon \in L(Q) \setminus L(G)$ because $(q, \varepsilon, S) \vdash^* (q, \varepsilon, S)$.

Which of the following statements is true for $G = (\{ S \}, \{ 0, 1 \}, (S \to 0S \mid 1), S)$?

1. $L(G) = L(Q)$.
2. $L(G) = N(Q)$.

$$
\begin{aligned}
\delta(q, \varepsilon, S) &= \{ (q, 0S), (q, 1) \} \\
\delta(q, 0, 0) &= \{ (q, \varepsilon) \} \\
\delta(q, 1, 1) &= \{ (q, \varepsilon) \} \\
\delta(q, \_, \_) &= \emptyset
\end{aligned}
$$

2. Yes.

Turing machines

Turing machines

- Simple computers.
- An idealised model of what it means to "compute".

Intuitive idea

- A tape that extends arbitrarily far in both directions.
- The tape is divided into squares.
- The squares can be blank or contain symbols, chosen from a finite alphabet.
- A read/write head, positioned over one square.
- The head can move from one square to an adjacent one.
- Rules that explain what the head does.

Rules

- A finite set of states.
- When the head reads a symbol
  (blank squares correspond to a special symbol):
    - Check if the current state contains a matching rule, with:
        - A symbol to write.
        - A direction to move in.
        - A state to switch to.
    - If not, halt.

The Church-Turing thesis

- Turing motivated his design partly by reference to what a human computer does.
- The Church-Turing thesis:
  Every effectively calculable function
  on the positive integers can be computed
  using a Turing machine.
- "Effectively calculable function" is not a well-defined concept, so this is not a theorem.

Syntax

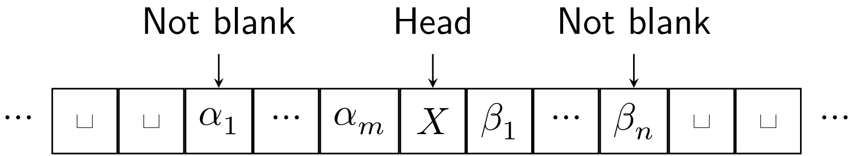
A Turing machine (TM) can be given as a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$:

- A finite set of states (Q).
- An input alphabet (Σ).
- A tape alphabet (Γ with $\Sigma \subseteq \Gamma$).
- A (partial) transition function ($\delta \in Q \times \Gamma \rightharpoonup Q \times \Gamma \times \{ \mathsf{L}, \mathsf{R} \}$).
- A start state ($q_0 \in Q$).
- A blank symbol ($\sqcup \in \Gamma \setminus \Sigma$).
- A set of accepting states ($F \subseteq Q$).

Instantaneous descriptions

An *instantaneous description* (ID) for a given TM is a 4-tuple (α, q, X, β), often written $\alpha q X \beta$:

- The current state ($q \in Q$).
- The symbol under the head ($X \in \Gamma$).
- Parts of the remaining tape ($\alpha, \beta \in \Gamma^*$).



Transition relation

The following relation between IDs defines what kinds of transitions are possible:

$$\frac{\delta(p, X) = (q, Y, \mathsf{R})}{(\alpha, p, X, Z\beta) \vdash (l(\alpha Y), q, Z, \beta)}$$

$$\frac{\delta(p, X) = (q, Y, \mathsf{R})}{(\alpha, p, X, \varepsilon) \vdash (l(\alpha Y), q, \sqcup, \varepsilon)}$$

The function l removes leading blanks.

Transition relation

$$\frac{\delta(p, X) = (q, Y, \mathsf{L})}{(\alpha Z, p, X, \beta) \vdash (\alpha, q, Z, r(Y\beta))}$$

$$\frac{\delta(p, X) = (q, Y, \mathsf{L})}{(\varepsilon, p, X, \beta) \vdash (\varepsilon, q, \sqcup, r(Y\beta))}$$

The function r removes trailing blanks.

Transition relation

The reflexive transitive closure of $\vdash$ can be defined inductively:

$$\frac{}{I \vdash^* I} \qquad\qquad \frac{I \vdash J \quad J \vdash^* K}{I \vdash^* K}$$

Transition relation

- A Turing machine:

$$(\{\,p\,\},\{\,0\,\},\{\,0,1,\sqcup\,\},\delta,p,\sqcup,\emptyset)$$

$$\delta(p,0)=(p,1,\mathsf{R})$$

- Some possible transitions:

$$p000 \vdash 1p00 \vdash 11p0 \vdash 111p$$

Consider the TM $M = (\{ p, q \}, \{ 0, 1 \}, \{ 0, 1, \sqcup \}, \delta, p, \sqcup, \emptyset)$, where δ is defined in the following way:

$$\delta(p, \sqcup) = (q, \sqcup, \mathsf{L})$$

$$\delta(p, 0) = (p, 1, \mathsf{R}) \qquad \delta(q, 0) = (q, 0, \mathsf{L})$$

$$\delta(p, 1) = (p, 0, \mathsf{R}) \qquad \delta(q, 1) = (q, 1, \mathsf{L})$$

Which of the following statements are true for M?

1. $p01 \vdash^* 10p_\sqcup$
2. $p01 \vdash^* q_\sqcup 10$
3. $p01 \vdash^* q_{\sqcup\sqcup}10$
4. $p111 \vdash^* 00p1$
5. $p111 \vdash^* 00q1$
6. $p111 \vdash^* 0q00$

Respond at `https://pingo.coactum.de/729558`.

$$\delta(p, \sqcup) = (q, \sqcup, \mathsf{L})$$

$$\delta(p, 0) = (p, 1, \mathsf{R}) \qquad \delta(q, 0) = (q, 0, \mathsf{L})$$

$$\delta(p, 1) = (p, 0, \mathsf{R}) \qquad \delta(q, 1) = (q, 1, \mathsf{L})$$

$$p01 \vdash 1p1 \vdash 10p_\sqcup \vdash 1q0 \vdash q10 \vdash q_\sqcup 10$$

$$p111 \vdash 0p11 \vdash 00p1 \vdash 000p_\sqcup \vdash 00q0 \vdash 0q00 \vdash$$
$$q000 \vdash q_\sqcup 000$$

Language

The language of a TM:

$$L((Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)) = \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} q \in F, X \in \Gamma, \alpha, \beta \in \Gamma^*, \\ q_0 w \vdash^* \alpha q X \beta \end{array} \right\}$$

(Here $q_0\varepsilon$ means $q_0\sqcup$.)

Halting

- Turing machines can fail to halt ($I_0 \vdash I_1 \vdash \dots$).
- A language is called *recursively enumerable* if it is the language of some Turing machine.
- A language is called *recursive* if it is the language of some Turing machine that always halts.
- There are languages that are recursively enumerable but not recursive.
- An example: The language of (strings representing) Turing machines that halt when given the empty string as input.

A hierarchy

A hierarchy of languages over the alphabet Σ (if $|\Sigma| \geq 2$):

$$\begin{array}{ll} \mathrm{Fin}(\Sigma^*) & \subsetneq \\ \text{Regular} & \subsetneq \\ \text{Context-free} & \subsetneq \\ \text{Recursive} & \subsetneq \\ \text{Recursively enumerable} & \subsetneq \\ \wp(\Sigma^*) & \end{array}$$

Some undecidable problems

The following things cannot, in general, be determined (using, say, a Turing machine that always halts):

- If a Turing machine halts for a given input.
- If two Turing machines accept the same language.
- ...

Consider the TM $M = (\{ p, q, r \}, \{ 1 \}, \{ 1, \sqcup \}, \delta, p, \sqcup, \{ r \})$, where δ is defined in the following way:

$$\delta(p, \sqcup) = (r, \sqcup, \mathsf{R})$$
$$\delta(p, 1) = (q, \sqcup, \mathsf{R})$$
$$\delta(q, 1) = (p, \sqcup, \mathsf{R})$$

Which of the following strings are members of $L(M)$? Does M always halt?

1. ε
2. 1
3. 11
4. 111
5. 1111
6. It always halts

Respond at `https://pingo.coactum.de/729558`.

Today

- Pushdown automata.
- Turing machines.

Next lecture

- A summary of the course.

Finite automata and formal languages (DIT323, TMV029)

Nils Anders Danielsson

2024-03-04

Today

- A summary of the course.

Proofs and induction

Proofs

Throughout the course we have talked about how one can prove various things.

Some basic proof methods

- To prove $p \Rightarrow q$, assume p and prove q.
- To prove $\forall x \in A.\ P(x)$, assume that we have an $x \in A$ and prove $P(x)$.
- To prove $p \Leftrightarrow q$, prove both $p \Rightarrow q$ and $q \Rightarrow p$.
- To prove $\neg p$, assume p and derive a contradiction.
- To prove $(p \Rightarrow q) \Rightarrow r$, assume that you are given a method for proving q given p, and use that to prove r.

Proofs as data

We talked about how some proofs can be seen as "data". Some examples:

- $\alpha \Rightarrow^* \beta$.
- $w \in L(G, S)$.

Proofs as games

We talked about how some proofs can be seen as games.

A game corresponding to
$\forall x \in A.\ \exists y \in P(x).\ Q(x, y) \Rightarrow R(x, y)$:

1. An adversary gives you $x \in A$.
   (If the adversary fails, then you win.)
2. Construct $y \in P(x)$ and give to the adversary.
3. The adversary gives you a proof of $Q(x, y)$.
4. You win if you can prove $R(x, y)$.

If you have a strategy that ensures that you always win, no matter what the adversary does, then the statement is true.

Induction

- Mathematical induction.
- Complete induction.
- Mutual induction.
- Inductively defined sets:
  - Primitive recursion.
  - Structural induction.
- Inductively defined subsets.

Induction

- Mathematical induction.

$$P(0) \wedge (\forall n \in \mathbb{N}.\ P(n) \Rightarrow P(n+1)) \Rightarrow \\ \forall n \in \mathbb{N}.\ P(n)$$

- Complete induction.
- Mutual induction.
- Inductively defined sets:
  - Primitive recursion.
  - Structural induction.
- Inductively defined subsets.

Induction

- Mathematical induction.
- Complete induction.

$$(\forall n \in \mathbb{N}.\ (\forall i \in \mathbb{N}.\ i < n \Rightarrow P(i)) \Rightarrow P(n)) \Rightarrow \forall n \in \mathbb{N}.\ P(n)$$

- Mutual induction.
- Inductively defined sets:
  - Primitive recursion.
  - Structural induction.
- Inductively defined subsets.

One way to structure a proof by induction

If you want to prove something by induction on the structure of a list of natural numbers:

- State what you want to prove, and how you intend to prove it:

  Let us prove $\forall xs \in List(\mathbb{N}).P(xs)$, where $P(xs) = \ldots$, by induction on the structure of the list.

- Prove each case:

  We have two cases:

  - $P(\text{nil})$ holds because...
  - Given $x \in \mathbb{N}$, $xs \in List(\mathbb{N})$ and $P(xs)$, we can prove $P(\text{cons}(x, xs))$ by...

Regular languages

Automata

Terminology, notation:

- Alphabets.
- Strings.
- Languages.
- Concatenation.
- Exponentiation.
- Kleene star.
- ...

DFAs

- Deterministic.
- 5-tuples.
- Transition diagrams.
- Transition tables.
- Transition functions for strings ($\hat{\delta}$).
- The language of a DFA.

DFAs

States can be:

- Accessible.
- Equivalent to each other.
- Distinguishable from each other.

NFAs

- Nondeterministic.
- 5-tuples.
- Transition diagrams.
- Transition tables.
- Transition functions for strings ($\hat{\delta}$).
- The language of an NFA.

DFAs and NFAs

- DFAs can easily be turned into NFAs.
- NFAs can be turned into DFAs:
  - The subset construction.
  - Optimisation: Skip inaccessible states.
  - Potential problem: Exponential blowup.

ε-NFAs

- Nondeterministic and with ε-transitions.
- 5-tuples.
- Transition diagrams.
- Transition tables.
- ε-closure.
- Transition functions for strings ($\hat{\delta}$).
- The language of an ε-NFA.

DFAs, NFAs and ε-NFAs

- NFAs can easily be turned into ε-NFAs.
- ε-NFAs can be turned into DFAs:
  - The subset construction with ε-closure.
  - Optimisation: Skip inaccessible states.

Regular expressions

- Syntax.
- The language of a regular expression.

Regular expressions

- From the first lecture (almost):

```
M-x replace-regexp RET
  add(\([^,]*\),\([^)]*\)) RET
  \1 + \2 RET
```

- We used regular expressions to define languages.
- Here a kind of regular expression is used to replace certain (sub)strings with other text.

Discuss whether it is a good idea to use this command.

```
M-x replace-regexp RET
  add(\([^,]*\),\([^)]*\)) RET
  \1 + \2 RET
```

You may want to consider a CFG with the following productions:

$$E \to \texttt{add(}\ E\ \texttt{,}\ E\ \texttt{)} \mid \texttt{mul(}\ E\ \texttt{,}\ E\ \texttt{)} \mid 0 \mid 1 \mid \ldots \mid 9$$

Discuss whether it is a good idea to use this command.

```
M-x replace-regexp RET
  add(\([^,]*\),\([^)]*\)) RET
  \1 + \2 RET
```

You may want to consider a CFG with the following productions:

$$E \to \texttt{add(}\ E\ \texttt{,}\ E\ \texttt{)} \mid \texttt{mul(}\ E\ \texttt{,}\ E\ \texttt{)} \mid 0 \mid 1 \mid \ldots \mid 9$$

Perhaps not intended:

$$\texttt{add(mul(1,2),3)} \mapsto \texttt{mul(1 + 2,3)}$$

Regular expressions

Proving that two regular expressions denote the same language:

- Use known equalities and equational reasoning.
- Use the semantics and antisymmetry:

$$L(e_1) \subseteq L(e_2) \land L(e_2) \subseteq L(e_1) \Rightarrow \\ L(e_1) = L(e_2)$$

- Replace variables with fresh symbols.
- Convert to DFAs. (There is an algorithm for checking if two DFAs denote the same language.)

ε-NFAs and regular expressions

Translating regular expressions to equivalent ε-NFAs:

- Easy.

Translating ε-NFAs to equivalent regular expressions:

- By eliminating states.
- By using Arden's lemma:
  The equation $X = AX \cup B$ has the least solution $X = A^*B$.

Regular languages

- Definition in terms of DFAs, NFAs, ε-NFAs or regular expressions.
- The pumping lemma.
- Closure properties:
  - Union.
  - Concatenation.
  - Kleene star/plus.
  - Intersection (product construction).
  - Complement.

The pumping lemma

For every alphabet Σ and *regular* language $L \subseteq \Sigma^*$.

$$\exists m \in \mathbb{N}.$$
$$\forall w \in L.\ |w| \geq m \Rightarrow$$
$$\exists t, u, v \in \Sigma^*.$$
$$w = tuv \wedge u \neq \varepsilon \wedge |tu| \leq m \wedge$$
$$\forall n \in \mathbb{N}.\ tu^n v \in L$$

- The pumping lemma can be used to prove that a language is not regular.

The pumping lemma

For every alphabet Σ and *regular* language $L \subseteq \Sigma^*$.

$$
\begin{array}{l}
\exists m \in \mathbb{N}. \\
\quad \forall w \in L.\ |w| \geq m \Rightarrow \\
\quad\quad \exists t, u, v \in \Sigma^*. \\
\quad\quad\quad w = tuv \wedge u \neq \varepsilon \wedge |tu| \leq m \wedge \\
\quad\quad\quad \forall n \in \mathbb{N}.\ tu^n v \in L
\end{array}
$$

- The last five lines are a necessary, but not a sufficient, condition for being regular:
  there is at least one non-regular language for which they hold.

The pumping lemma

For every alphabet Σ and *regular* language $L \subseteq \Sigma^*$.

$$
\begin{array}{l}
\exists m \in \mathbb{N}. \\
\quad \forall w \in L.\ |w| \geq m \Rightarrow \\
\quad\quad \exists t, u, v \in \Sigma^*. \\
\quad\quad\quad w = tuv \wedge u \neq \varepsilon \wedge |tu| \leq m \wedge \\
\quad\quad\quad \forall n \in \mathbb{N}.\ tu^n v \in L
\end{array}
$$

- Do not give "the pumping lemma holds, so the language is regular" as an exam answer.

Regular languages

Algorithms:

- Conversions between different formats.
- Is the language empty?
- Is a given string a member of the language?
- Are two regular languages equal?
  - Are two states equivalent?
- Minimisation of DFAs.

Context-free languages

Context-free grammars

4-tuples:

- Nonterminals.
- Terminals.
- Productions.
- Start symbol.

Context-free grammars

The language of a CFG can be defined in several equivalent ways:

- Derivations.
- Leftmost (rightmost) derivations.
- Parse trees.
- Recursive inference.

Context-free grammars

- Ambiguous grammars.
- Associativity.
- Precedence.

Context-free grammars

- Chomsky normal form:
  $A \to a$ or $A \to BC$.
- $\textsc{Bin}$, $\textsc{Del}$, $\textsc{Unit}$, $\textsc{Term}$.

Pushdown automata

- A kind of finite automaton with a single stack.
- 7-tuples.
- Instantaneous descriptions.
- Transition relation ($\vdash$).
- The languages of a PDA P: $L(P)$ and $N(P)$.

Context-free languages

- Definition in terms of CFGs or PDAs, which define the same class of languages.
- The pumping lemma.
- Closure properties:
  - Substitution.
  - Union.
  - Concatenation.
  - Kleene star/plus.
  - Homomorphism.
  - Intersection with a regular language.

Which of the following languages, if any, are context-free? Try to use closure properties.

1. $\{uuvv \mid u \in \{0\}^+, v \in \{1\}^+\} \cup \{uvvu \mid u \in \{0\}^+, v \in \{1\}^+\}$
2. $\{uuvv \mid u \in \{0\}^+, v \in \{1\}^+\} \cap \{uvvu \mid u \in \{0\}^+, v \in \{1\}^+\}$
3. $\{ssttuvvu \mid s, u \in \{0\}^+, t, v \in \{1\}^+\}$
4. $\{uuvvuvvu \mid u \in \{0\}^+, v \in \{1\}^+\}$
5. $\{(uvvu)^n \mid u \in \{0\}^+, v \in \{1\}^+, n \in \mathbb{N}\}$
6. $\{uvu \mid u \in \{0,1\}^*, v \in \{2,3\}^*\}$

Respond at `https://pingo.coactum.de/729558`.

Which of the following languages, if any, are context-free? Try to use closure properties.

1. $\{uuvv \mid u \in \{0\}^+, v \in \{1\}^+\} \cup$
   $\{uvvu \mid u \in \{0\}^+, v \in \{1\}^+\}$

Yes. The union of two context-free languages.

Which of the following languages, if any, are context-free? Try to use closure properties.

2. $\{uuvv \mid u \in \{0\}^+, v \in \{1\}^+\} \cap$
   $\{uvvu \mid u \in \{0\}^+, v \in \{1\}^+\}$

Yes. The intersection of a context-free language and a regular language.

Which of the following languages, if any, are context-free? Try to use closure properties.

3. $\{ssttuvvu \mid s, u \in \{0\}^+, t, v \in \{1\}^+\}$

Yes. The concatenation of two context-free languages.

Which of the following languages, if any, are context-free? Try to use closure properties.

4. $\{uuvvuvvu \mid u \in \{0\}^+, v \in \{1\}^+\}$

No. This is $\{ 0^{2m}1^{2n}0^m1^{2n}0^m \mid m, n \in \mathbb{N} \setminus \{ 0 \} \}$. Use the pumping lemma.

Which of the following languages, if any, are context-free? Try to use closure properties.

5. $\{(uvvu)^n \mid u \in \{0\}^+, v \in \{1\}^+, n \in \mathbb{N}\}$

No. Denote the language by L.
Note that $L \neq \{uvvu \mid u \in \{0\}^+, v \in \{1\}^+\}^*$.
If L had been context-free, then the language

$$
\begin{aligned}
&L \cap L(0^+1^+0^+1^+0^+) = \\
&\{uvvuuvvu \mid u \in \{0\}^+, v \in \{1\}^+\} = \\
&\{0^m 1^{2n} 0^{2m} 1^{2n} 0^m \mid m, n \in \mathbb{N} \setminus \{\, 0 \,\}\}
\end{aligned}
$$

would have been context-free, but it is not (use the pumping lemma).

Which of the following languages, if any, are context-free? Try to use closure properties.

6. $\{uvu \mid u \in \{0,1\}^*, v \in \{2,3\}^*\}$

No, because if this language is context-free, then the intersection of this language with $\{ 0, 1 \}^*$ is context-free, and that language is $\{uu \mid u \in \{ 0, 1 \}^*\}$, which is not context-free.

Context-free languages

Algorithms:

- Generating symbols.
- Is the language empty?
- Nullable nonterminals.
- Is the empty string a member of the language?
- Is a nonempty string a member of the language?
  - The CYK algorithm.

Recursive or recursively enumerable languages

Turing machines

- A kind of simple computer.
- Read/write head, unbounded tape, finite set of states.
- 7-tuples.
- Instantaneous descriptions.
- Transition relation ($\vdash$).
- The language of a TM.
- Halting.
- Undecidable problems.

Recursive languages

- Definition in terms of (halting) TMs, or lambda expressions, or recursive functions, or...
- The Church-Turing thesis.

Recursively enumerable languages

- Definition in terms of TMs, or lambda expressions, or recursive functions, or...

A hierarchy

A hierarchy of languages over the alphabet Σ (if $|\Sigma| \geq 2$):

$$
\begin{array}{ll}
\text{Finite} & \subsetneq \\
\text{Regular} & \subsetneq \\
\text{Context-free} & \subsetneq \\
\text{Recursive} & \subsetneq \\
\text{Recursively enumerable} & \subsetneq \\
\wp(\Sigma^*) &
\end{array}
$$

A hierarchy

A hierarchy of languages over the alphabet Σ (if $|\Sigma| \geq 2$):

$$\begin{array}{ll} \text{Finite} & \subsetneq \\ \text{Regular} & \subsetneq \\ \text{Context-free} & \subsetneq \\ \text{Recursive} & \subsetneq \\ \text{Recursively enumerable} & \subsetneq \\ \wp(\Sigma^*) & \end{array}$$

It might not be a good idea to give "the language is context-free, but not regular" as an exam answer.

Discuss what you have learnt in this course.

- What has been most interesting?
- What has been least interesting?
- What would you like to know more about?
- ...

Coming up

- Next lecture: Old exam questions.
  Perhaps the following ones:
  - 2020-03-19: 2, 5 and 6.
  - 2021-03-18: 2.
  - 2021-08-18: 1, 2 and 4.
- Deadline for the seventh assignment: **Friday**.