Main Points of the Course

What has been covered: chapters 1 to 5 + 7

Plus abstract states/Myhill-Nerode

Mathematical Definitions

You should know what are, mathematically, DFA, NFA ϵ-NFA, CFG

For instance, a NFA is $(Q, \Sigma, q_0, \delta, F)$ where Q is a finite state (set of states), Σ a finite set (alphabet), $q_0 \in Q$,

$$\delta : Q \times \Sigma \rightarrow Pow(Q)$$

and $F \subseteq Q$

Another view of NFA is *labelled transition system*

Mathematical Definitions

You should know also what is a regular expression

Given a regular expression E, what is the language $L(E)$ represented by E

Constructions on FA

The 3 main constructions

1. product of two DFAs (or NFAs), to compute union, intersection of regular languages

2. subset construction NFA $\longrightarrow$ DFA

3. minimization DFA $\longrightarrow$ DFA (does *not* work for NFA!!)

Constructions on FA

Some other constructions we have seen

Complement of a language: complement of an automaton (this works *only* for DFA)

Reverse of a language: reverse of an automaton (work for DFA and NFA; we may get a NFA even if we start with a DFA)

Be *careful*: given E_1, E_2 we can compute E such that $L(E) = L(E_1) \cap L(E_2)$ but $E_1 \cap E_2$ is not a regular expression (only in a generalised sense)

Similarly given E_1 we can compute E such that $L(E)$ is $\overline{L(E)}$ the complement of $L(E_1)$ but $\overline{E}$ is *not* a regular expression

From FA to regular expressions

FA $\rightarrow$ regular expression

We have 3 methods to compute a regular expression E such that $L(E) = L(A)$

1. method similar to Warshall's algorithm: section 3.2.1
2. eliminating states: section 3.2.2
3. writing a system of equations, and method of successive elimination

From FA to CFG

It is direct to associate a CFG to a ϵ-NFA

$$S_0 \rightarrow S_1 \mid +S_1 \mid -S_1 \qquad S_1 \rightarrow dS_1 \mid dS_4 \mid \cdot S_2$$

$$S_2 \rightarrow dS_3 \qquad S_3 \rightarrow \epsilon \mid dS_3 \qquad S_4 \rightarrow \cdot S_3$$

$$d \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

From regular expressions to FA

regular expression $\longrightarrow$ ϵ-NFA

ϵ-NFA $\longrightarrow$ NFA

NFA $\longrightarrow$ DFA (subset construction)

From regular expressions to FA

Other more direct approach with abstract states

Example: $0(10)^*$

Regular expressions

Basic equalities on regular expressions, like

$$E(F+G) = EF+EG \qquad (ac)^*a = a(ca)^*$$

For instance, nice solution to $(ab+a)^*a = a(ba+a)^*$

$$(ab+a)^*a = (a(b+\epsilon))^*a = a((b+\epsilon)a)^* = a(ba+a)^*$$

In practice: try to see what are the possible “first” elements in each languages when trying to decide if two languages are equal. (Good exercise: program in Haskell an equality test)

Minimization

Table-filling algorithm well-described in section 4.4.3

Does not work for NFA

You should know that it is uniquely defined: if $L(A_1) = L(A_2)$ and A_1, A_2 are minimal then A_1 and A_2 are identical (up to renaming of states), and the states are the abstract states

Non Regular Languages

Intuitively: a language is non regular when unbounded amount of memory is needed for a machine to recognize it

Typical example

$$S \to aSb \mid \epsilon$$

One proves by an argument *by contradiction*, using the *pigeon-hole principle* (see page 66) that a finite-state machine cannot recognize $L(G)$

Section 4.1

Another approach: $L(G)$ has infinitely many abstract states

Regular and Context-Free Languages

For regular languages: you should now how to decide

$$L(A) \neq \emptyset \qquad w \in L(A) \qquad L(A_1) \subseteq L(A_2)$$

For context-free languages, you should know how to decide

$$L(G) \neq \emptyset$$

There is no algorithm for $L(G_1) \subseteq L(G_2)$

No algorithm to compute if G is *ambiguous* (see section 9.5)

Regular and Context-Free Languages

How to decide

$$L(G) \neq \emptyset$$

if G is the grammar

$$S \rightarrow aB \mid BC \qquad A \rightarrow aA \mid c \mid aDb$$

$$B \rightarrow DB \mid C \qquad C \rightarrow b \mid B$$

we compute the *generating* symbols

You should know also how to compute the *accessible* or *reachable* symbols

Induction on length of derivations

Consider the following grammar G

$$S \rightarrow \ a \mid b \mid SSS$$

Show that $L(G)$ is the set of all words in $\{a,b\}^*$ of *odd* length.

$L = L(G)$ is inductively defined by the clauses

- $a, b \in L$
- if $w_1, w_2, w_3 \in L$ then $w_1 w_2 w_3 \in L$

Contex-Free Languages

Let M be the set of words of odd length.

We prove $L = M$ by proving $L \subseteq M$ *and* $M \subseteq L$

$L \subseteq M$ can be proved by induction on the length of $S \Rightarrow^* w$:

- $S \Rightarrow a,\ S \Rightarrow b$ are of length 1, hence $a, b \in M$
- if $S \Rightarrow SSS \Rightarrow^* w_1w_2w_3$. By induction $|w_i|$ is odd and so is $|w_1w_2w_3|$

Context-Free Languages

We have also to prove $M \subseteq L$

We prove $w \in M$ implies $w \in L$ by induction on $|w|$

If $|w| = 1$ then $w = a$ or b

If $|w| > 1$ then $w = c_1c_2w'$ with $c_i = a$ or b. We know $w' \in L$ by induction hypothesis. Also, $a, b \in L$. Hence $w \in L$

Contex-Free Languages

Consider the following grammar G

$$S \rightarrow \ A1B \qquad A \rightarrow 0A \mid \epsilon \qquad B \rightarrow 1B \mid \epsilon$$

Show that G is *not* ambiguous

There is *no* general method to solve this kind of problem (section 9.5)

First we try to understand what is $L(G)$

Here $L(G) = L(0^*11^*)$

Context-Free Languages

We show that if $w \in L(G)$ then w has a *unique* leftmost derivation by induction on $|w|$

Context-Free Languages

We do a case analysis if w starts with the symbol 0 or not

If $w = 0w'$ then the leftmost derivation has to start

$$S \Rightarrow_{lm} A1B \Rightarrow_{lm} 0A1B$$

with a leftmost derivation of

$$A1B \Rightarrow_{lm}^{*} w'$$

We know by induction hypothesis that w' has a unique leftmost derivation

$$S \Rightarrow_{lm} A1B \Rightarrow_{lm}^{*} w'$$

Context-Free Languages

If $w = 1w'$ then $w' = 1^n$ the leftmost derivation has to start

$$S \Rightarrow_{lm} A1B \Rightarrow_{lm} 1B$$

with a leftmost derivation of

$$B \Rightarrow_{lm}^{*} w'$$

We show by induction on n that there is a unique leftmost derivation

$$B \Rightarrow_{lm}^{*} 1^n$$

Variation on Automata: Pushdown Automata

Not seen in the course

NFA + stack = context-free language

A stack is needed for recognizing a language such as

$$S \to \epsilon \mid aSb$$

Variation on Automata: Pushdown Automata

DFA $+$ stack is less powerful

inclusion $L(A_1) \subseteq L(A_2)$ decidable for this fragment (proved in 1998!!)

There is no algorithms for testing $L(G_1) \subseteq L(G_2)$ and so no algorithm for $L(A_1) \subseteq L(A_2)$, if A_i NFA with stacks

Variation on Automata: Turing Machines

DFA + tape

The machine can write also on the tape

All *recursive* languages

Strict hierarchy between languages:

regular $\subset$ context-free $\subset$ recursive

With two stacks we get the same languages as recursive languages. See section 8.2