Consolidation

Survey results

Which topics do you (presumably) feel most comfortable with?

1. Deterministic finite automata (97%)
2. Non-deterministic finite automata (88%)
3. Regular expression (88%)
4. Equivalence of partial DFAs (82%)
5. Minimisation of partial DFAs (82%)
6. Converting regular expressions to DFAs (62%)

64185902

Survey results

Which topics do you most want to revise?

1. Proving non-regularity of a language (79%)
2. Chomsky normal form (68%)
3. Ambiguity of context-free grammars (65%)
4. Context-free languages (53%)
5. Complexity (53%)
6. Introduction to Turing Machines (44%)

Of course, this doesn't cover everything for everyone, so please come to the exercises classes and office hours this week so that we can help you further!

64185902

Proving non-regularity of a language

Recall from MLFCS that a set is either *finite* or *infinite*.

For example, the empty set, the set $\{5, 4, 3, 2, 1\}$, and the set of everyone in this room are finite; whereas the sets $\mathbb{N}$ and $\mathbb{R}$ are infinite.

The set $\mathbb{N}$ is *countably infinite*, while $\mathbb{R}$ is uncountable. A set hat is either finite or countably infinite is called *countable*.

64185902

Proving non-regularity of a language

Regular languages are those that can be matched by a regular expression or (by Kleene's theorem) a deterministic finite automata.

Given any (non-empty) alphabet, there are only a *countable infinity* of possible regular expressions over that alphabet and – furthermore – a deterministic finite automata must have a *finite* number of states.

But there are an uncountable number of languages in total, meaning that *not all languages are regular*.

64185902

Proving non-regularity of a language

An example of a non-regular language is the set of well-bracketed words over the language $\{(,)\}$.

For example, ()((())) is well-bracketed, while (()(()) isn't.

64185902

Proving non-regularity of a language

We want to *prove* that this language is non-regular, so that we know there is no regex or DFA that accepts it.

How do we do that?

64185902

Proving non-regularity of a language

We start our proof by assuming that the language *is* regular, meaning by Kleene's theorem that there *is* a DFA that accepts it.

Then, we just need to show that there is a problem with this DFA – namely, that it would need an infinite number of states! That would mean the DFA cannot actually exist, and therefore the language is not regular.

64185902

Proving non-regularity of a language

So how can we show that our assumed DFA does not exist?

By showing that, for a countably-infinite subset $N \subseteq \mathbb{N}$ (usually $\mathbb{N}$ itself), for each natural number $n \in N$, there is a state x_n that is not equivalent to any other state.

This is usually done by showing that for all $n, m \in N$ such that $n \neq m$ or some weaker condition that means they are different (e.g. $n < m$) there are states x_n and x_m, one of which accepts a word and one of which rejects it.

By showing this, we have shown that there is an infinite number of non-equivalent states, contradicting the existence of the assumed deterministic *finite* automata.

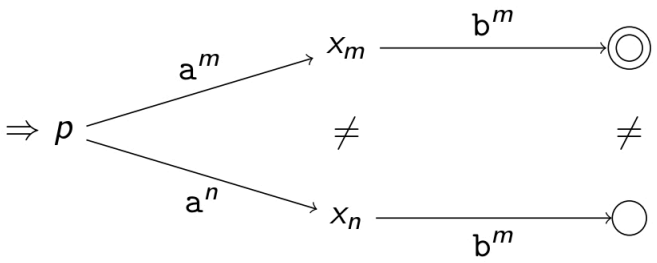
64185902

Proving non-regularity of a language

From the previous slide:

> "Show that for all $n, m \in N$ such that $n \neq m$ or some weaker condition that means they are different (e.g. $n < m$) there are states x_n and x_m, one of which accepts a word and one of which rejects it."

The previous slide explained with a diagram: $\forall n, m \in \mathbb{N}.n \neq m$.



64185902

Proving non-regularity of a language

Let's prove that the language of well-bracketed words L is non-regular:

- Suppose L is regular; then there's a DFA that recognises it,
- Let's say that, for every $n \in \mathbb{N}$, x_n is the state reached by reading n-many open brackets (,
  - Thus, x_0 is the initial state.
- Given $n, m \in \mathbb{N}$ such that $n < m$, the state x_m accepts the word $)^m$ but x_n rejects the word $)^m$.
- Therefore, x_n and x_m are not equivalent, meaning there is an infinite number of states. This contradicts the finiteness of the assumed DFA, meaning that it cannot exist.
- Therefore, as there is no DFA that recognises the language, the language is non-regular.

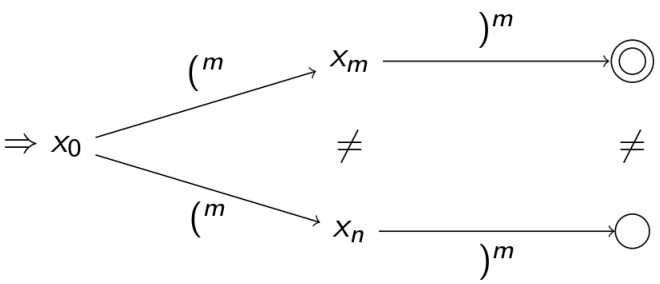
64185902

Proving non-regularity of a language

From the previous slide:

> "Given $n, m \in \mathbb{N}$ such that $n < m$, the state x_m accepts the word $)^m$ but x_n rejects the word $)^m$. Therefore, x_n and x_m are not equivalent."

The previous slide explained with a diagram: $\forall n, m \in \mathbb{N}.n < m$.



64185902

Context-free languages

Regular languages can be described using regular expressions and deterministic finite automata. But, as we saw with the language of well-bracketed expressions, there are non-regular languages too.

We will now explore *context-free languages*, which are languages that can be described using *context-free grammars* (CFGs).

The set of context-free languages is a strict super-set of the set of regular languages: this means that:

- Every regular language is a context-free language,
- There are more context-free languages than regular languages (i.e. there are languages that can be described by a CFG that can't be described by a regex/DFA).

Context-free languages

A CFG is consists of a set of *non-terminals* or *variables* (one of which is the *start variable*), a set of *terminals* and a set of *rules* from a single variable to several possible strings consisting of variables or a terminals.

For example, in the CFG

$$\begin{aligned} \Rightarrow A &::= 3 \mid 5 \mid A + A \mid A \times A \mid (A) \mid \text{if } B \text{ then } A \text{ else } A \\ B &::= A > A \mid B \text{ and } B \mid (B) \end{aligned}$$

What are the variables, start variable, terminals and rules?

- The variables are A and B, with A being the start variable,
- The terminals are 3 and 5,
- The rules are written together, e.g. $A ::= A \times A$.

Context-free languages

We can *derive* words from the grammar by beginning at the start variable and repeatedly applying rules to one variable at a time until we have the desired word.

For example, we can derive the word $\text{if } 3 > (3 + 5) \text{ then } 3 \text{ else } 5$ from the grammar on the previous slide as follows:

$$\begin{aligned}
\dot{A} &\rightsquigarrow \text{if } B \text{ then } \dot{A} \text{ else } A \\
&\rightsquigarrow \text{if } B \text{ then } 3 \text{ else } \dot{A} \\
&\rightsquigarrow \text{if } \dot{B} \text{ then } 3 \text{ else } 5 \\
&\rightsquigarrow \text{if } A > \dot{A} \text{ then } 3 \text{ else } 5 \\
&\rightsquigarrow \text{if } \dot{A} > (A) \text{ then } 3 \text{ else } 5 \\
&\rightsquigarrow \text{if } 3 > (\dot{A}) \text{ then } 3 \text{ else } 5 \\
&\rightsquigarrow \text{if } 3 > (A + \dot{A}) \text{ then } 3 \text{ else } 5 \\
&\rightsquigarrow \text{if } 3 > (\dot{A} + 5) \text{ then } 3 \text{ else } 5 \\
&\rightsquigarrow \text{if } 3 > (3 + 5) \text{ then } 3 \text{ else } 5
\end{aligned}$$

Context-free languages

$$\begin{aligned}
\Rightarrow A &::= 3 \mid 5 \mid A + A \mid A \times A \mid (A) \mid \text{if } B \text{ then } A \text{ else } A \\
B &::= A > A \mid B \text{ and } B \mid (B)
\end{aligned}$$

$$\begin{aligned}
\dot{A} &\rightsquigarrow \text{if } B \text{ then } \dot{A} \text{ else } A \\
&\rightsquigarrow \text{if } B \text{ then } 3 \text{ else } \dot{A} \\
&\rightsquigarrow \text{if } \dot{B} \text{ then } 3 \text{ else } 5 \\
&\ldots \\
&\rightsquigarrow \text{if } 3 > (\dot{A} + 5) \text{ then } 3 \text{ else } 5 \\
&\rightsquigarrow \text{if } 3 > (3 + 5) \text{ then } 3 \text{ else } 5
\end{aligned}$$

Note that we put a dot above the variable that we are about to replace – this makes it clear which rule we apply step-by-step!

Context-free languages

Some words have different derivations. For example, the following is the *leftmost derivation* of the previous example (leftmost means that, at each step, we perform a rule on the leftmost variable):

$$\begin{aligned}
\dot{A} &\rightsquigarrow \text{if } \dot{B} \text{ then } A \text{ else } A \\
&\rightsquigarrow \text{if } \dot{A} > A \text{ then } A \text{ else } A \\
&\rightsquigarrow \text{if } 3 > \dot{A} \text{ then } A \text{ else } A \\
&\rightsquigarrow \text{if } 3 > (\dot{A}) \text{ then } A \text{ else } A \\
&\rightsquigarrow \text{if } 3 > (\dot{A} + A) \text{ then } A \text{ else } A \\
&\rightsquigarrow \text{if } 3 > (3 + \dot{A}) \text{ then } A \text{ else } A \\
&\rightsquigarrow \text{if } 3 > (3 + 5) \text{ then } \dot{A} \text{ else } A \\
&\rightsquigarrow \text{if } 3 > (3 + 5) \text{ then } 3 \text{ else } \dot{A} \\
&\rightsquigarrow \text{if } 3 > (3 + 5) \text{ then } 3 \text{ else } 5
\end{aligned}$$

64185902

Context-free languages – Ambiguity

Derivations can also be represented using *derivation trees*.

When the *same word* can be *derived* in two (or more) *different ways* from the same CFG (i.e. there are at least two different derivation trees), we call the language generated from the grammar *ambiguous*.

For example, let's show that the following grammar is ambiguous (the alphabet is $\{\mathtt{a}, \mathtt{b}\}$):

$$\begin{aligned}
\Rightarrow P &::= \varepsilon \mid Q\mathtt{a} \mid \mathtt{a}Q \\
Q &::= \mathtt{aa}P \mid \mathtt{b}R \\
R &::= Q\mathtt{a}
\end{aligned}$$

64185902

Context-free languages – Chomsky Normal Form

Chomsky Normal Form (CNF) is a certain way of writing context-free grammars. Converting a CFG in CNF can be used in some cases to reduce the complexity of deriving a word from that grammar.

In CNF, we are only allowed rules of the following kind

$$\begin{aligned} A &::= BC \\ A &::= a \end{aligned}$$

where a is any terminal and A, B, and C are any variables – except that B and C may not be the start variable. In addition, there can be a rule $S ::= \varepsilon$, but only if S is the start variable.

Context-free languages – Chomsky Normal Form

To convert any given CFG into CNF, use the steps outlined below:

1. We begin by introducing a new start symbol (variable) to the grammar.
2. In the second step, we remove all of the ε-rules of the form $A ::= \varepsilon$.
3. In the third step, we remove all of the unit productions of the form $A ::= B$.
4. We may need to patch-up/fix the grammar to make sure that it still produces the original language.
5. In the end, we will convert the remaining rules into proper form.

Context-free languages

Let's convert the following example to CNF:

$$\begin{aligned} \Rightarrow A &::= BAB \mid B \mid \varepsilon \\ B &::= 00 \mid \varepsilon \end{aligned}$$

1. We begin by introducing a new start symbol (variable) to the grammar.
2. In the second step, we remove all of the ε-rules of the form $A ::= \varepsilon$.
3. In the third step, we remove all of the unit productions of the form $A ::= B$.
4. We may need to patch-up/fix the grammar to make sure that it still produces the original language.
5. In the end, we will convert the remaining rules into proper form.

64185902

Complexity

Consider a sample algorithm, e.g. sorting a list.

We want to be able to compute:

- The **worst-case complexity**: this gives us an *upper bound* on the cost. It is determined by the *most difficult* input and provides a guarantee for all inputs.
- The **average-case complexity**: this gives us the *expected cost* for a random input. It requires a model for *random input* and provides a way to predict performance.

64185902

Complexity

For a given algorithm or problem, we may want to compute the worst and average-case running times in terms of the size of the input.

For example, take the following program that operates on an array of characters that are all `a` or `b` or `c`.

```
void f (char[] p) {
  elapse(1 second);
  for (nat i = 0; i<p.length(), i++) {
    if (p[i]=='a') {
      elapse(1 second);
    } else {
      elapse(2 seconds);
    }
    elapse (1 second);
  }
}
```

64185902

Complexity

For an array of length 2, assuming `a`, `b`, `c` are equally likely and the characters are independent, we can compute the best and average running times....

64185902

Complexity

Array contents	Probability	Time	Probability × time
`aa`	$\frac{1}{9}$	$5s$	$\frac{5}{9}s$
`ab`	$\frac{1}{9}$	$6s$	$\frac{2}{3}s$
`ac`	$\frac{1}{9}$	$6s$	$\frac{2}{3}s$
`ba`	$\frac{1}{9}$	$6s$	$\frac{2}{3}s$
`bb`	$\frac{1}{9}$	$7s$	$\frac{7}{9}s$
`bc`	$\frac{1}{9}$	$7s$	$\frac{7}{9}s$
`ca`	$\frac{1}{9}$	$6s$	$\frac{2}{3}s$
`cb`	$\frac{1}{9}$	$7s$	$\frac{7}{9}s$
`cc`	$\frac{1}{9}$	$7s$	$\frac{7}{9}s$
Worst case: `bb`		$7s$	
Average case			$6\frac{1}{3}s$

64185902

Complexity

However, we don't usually work out the precise running time of a program; indeed we don't usually have the information needed to do so.

Instead we work out the *complexity*, which is a kind of rough estimate of the running time that ignores *small arguments* and *constant factors*.

- Why should we ignore small arguments?
- Why should we ignore constant factors?

Because when looking at the complexity of an algorithm, we're interested in how it *scales* to large values.

64185902

Complexity

That's why we use **Big O** notation to explain the complexity of an algorithm in a way which ignores small arguments and constant factors.

Let's say that $f \in (\mathbb{N} \to \mathbb{R}^+)$ is a function that gives the running time $f(n) \in \mathbb{R}^+$ of a program given an input of size $n \in \mathbb{N}$. We can explain the time complexity of f by finding a function $g \in (\mathbb{N} \to \mathbb{R}^+)$ that f is proportional to.

Once we've done this, we say that $f \in O(g)$, which means "f grows (at most) proportionally to g".

Complexity

So how do we *show* that $f \in O(g)$?

We need to show that $g(n)$ gives an *upper bound* for $f(n)$, *once small values and constant factors have been ingored*. To do this, we find two constants $M \in \mathbb{N}$ and $C \in \mathbb{N} - M$ is used to ensure we only consider sufficiently large values of n, while C accounts for constant scaling.

The idea is that f will never be more than $C * g(n)$, for values of n of size at least M:

$$\exists M.\exists C.\forall n \geq M.f(n) \leq Cg(n).$$

Complexity

We *show* that $f \in O(g)$ by proving

$$\exists M.\exists C.\forall n \geq M.f(n) \leq Cg(n).$$

Past Exam Question: Given two programs A and B, we build the program AB which first runs A and then B on the same input of length n. The worst case running time of A is given by a function $T(n)$, which is $n^7 + 5$ for $n < 1000$ and $n^3 + 5$ for $n \geq 1000$. The worst case running time of B is given by a function $R(n)$, which is $O(n \log_{10} n)$. Show carefully that the worst case running time of AB is $O(n^3)$. **[10 marks]**

64185902

Introduction to Turing Machines

As we've not finished Turing Machines yet, let's revisit Handout 5b in preparation for Week 7.

64185902