CS 360 Term Notes

Craig, Shale
sakcraig@uwaterloo.ca

February 25, 2014

Contents

Chapter 1

Introduction

1.1 Countable and Uncountable sets

When talking about the size of a set, they can be infinite or finite. When talking about an order, sets can be countable (i.e. enumerable) or uncountable (i.e. innumerable).

Iff some set A is enumerable, then there must exist an invertible function f such that $f : \mathbb{N} \to A$.

1.2 Alphabets, Strings, and Languages

Alphabets (usually denoted by Σ) are sets of unique characters. Strings are ordered sets of (possibly repeating characters in Σ). Languages are defined sets of words.

Chapter 2

Regular Languages

2.1 Regular Languages

Regular Languages are languages recognizable by Regular Expressions.

All finite languages are regular. Not all regular languages are finite.

2.2 State Machines

2.2.1 DFAs

DFA is short for Deterministic Finite State Machine.

A DFA A is a 5-tuple of $\langle Q, \Sigma, \delta, q_0, F \rangle$. Where Q is a set of possible states, Σ is the language, δ is a transfer function $\delta : Q, \Sigma \to Q$.

2.2.2 NFAs

DFA is short for Non-deterministic Finite State Machine.

A NFA A is a 5-tuple of $\langle Q, \Sigma, \delta, q_0, F \rangle$. Where Q is a set of possible states, Σ is the language, δ is a transfer function $\delta : Q, \Sigma \to P(Q)$. $P(Q)$ is the powerset of Q (i.e. the set of all subsets of Q).

2.3 Equivalence of NFAs and DFAs

NFAs and DFAs are equivalent.

We can change an DFA into a NFA trivially.

We can change a NFA into a DFA through the powerset construction - i.e. we can define δ_{DFA} by calling each $R \in P(Q_{NFA})$ a state of our new NFA. Transition functions will be $R, \Sigma \to Q_{DFA}$.

2.4 Regular Operations

Given two languages A and B, there are three regular operations:

1. $C = A \cup B$:

   C is defined to be the language that accepts any string in A or B or both.

2. $C = A \cap B$:

C is defined to be the language that accepts any string in both A and B.

3. $C = A^*$:

   C is defined to be the language that accepts any string in $\{\varepsilon\} \cup A \cup AA \cup AAA \cup \ldots$

   ε is the empty string.

2.5 Closure Properties of Regular Languages

Given that A and B are regular:

1. $A \cup B$ is regular
2. $A \cap B$ is regular
3. A^* is regular

2.6 Regular Expressions and Their Equivalence with NFAs and DFAs

Regular expressions describe the same languages as NFAs/DFAs:

Regular expressions can be turned into NFA's trivially. NFAs can describe the same languages as DFAs, so that reduction works.

Transitions in DFAs can be reduced to Regular Expressions by forcing transitions to have Regular Expressions, then removing state diagrams until a Regular Expression appears.

2.7 Techniques for Proving Certain Languages are Non-Regular

We can use a few techniques to prove certain languages are non-regular, but the main one seems to be that the pumping lemma holds for every regular language.

The pumping lemma for Regular languages states:

For every a Regular Language L, there is a $p \geq 1$ such that every $w \in L$ where $|w| \geq p$ can be written as $w = xyz$. Where the following holds:

- $|y| \geq 1$
- $|xy| \leq p$
- for all $i \geq 0$, $xy^iz \in L$

i.e. every Regular Language L has a pumping length p such that every string w where $|w| \geq p$ must have a portion that can be repeated indefinitely (or removed altogether).

2.8 Further Discussion of Regular Languages, Finite Automata, and Regular Expressions

TODO: Not sure what we did here.

Chapter 3

Context-Free Grammars and Languages

3.1 Context-Free Grammars and Languages

A context-free grammar G defines a context-free language $L(G)$.

CFG's are defined as a 4-tuple $G = (V, \Sigma, R, S)$, where:

- V is a finite set of non-terminal characters.
- Σ is a set of terminal characters that make up the language. As a rule, $V \cap \Sigma = \varnothing$.

- R is a relation $R : V \to (V \cup \Sigma)^*$.
- S is the start variable (or start symbol. By definition, $S \in V$.

For example, this is a context-free grammar that accepts all strings that are odd length and contain a 1 in the middle:

$$S \to USU|1$$
$$U \to 0|1$$

Every regular language can be expressed by a CFG.

3.2 Parse Trees and Ambiguity

Parse trees are simple, and they are ambiguous.

For example, how many ways are there to parse `x+++x`? It's probably defined in the language spec, but a CFG may mess up interpreting this line of code.

3.3 More Examples of Context-Free Grammars

I can't believe we spent a full class on this. It's simple.

3.4 Normal forms for CFGs

Chomsky Normal Form is a form of writing CFGs. All CFGs can be written in CNF.

A grammar is said to be in CNF if all production rules are in the form:

$$A \to BC$$
$$A \to \alpha$$
$$A \to \varepsilon$$

Where B and C are nonterminal symbols, are not S, α is a terminal symbol. Of course, ε is the empty string.

3.5 Closure Properties for CFLs

Context-Free Languages are closed under the following operations: (Assume that A, B are in CFL)

1. $A \cup B$
2. AB
3. reverse(A)
4. A^*

A, $A \cap B$, $A \setminus B$ are not CFLs.

3.6 Techniques for Proving Certain Languages are Non-Context-Free

We can use a few techniques to prove certain languages are non-Context-Free, but the main one is the pumping lemma holds for every Context-Free language.

The pumping lemma for Context-Free languages states:

For every a Context-Free Language L, there exists a $p \geq 1$ such that for every $w \in L$ where $|w| \geq p$ can be written as $w = uvxyz$. Where the following holds:

- $|vxy| \leq p$
- $vx \neq \varepsilon$
- For all $i \geq 0$, $uv^ixy^iz \in L$.

i.e. every Context-Free Language L has a pumping length p such that every string w where $|w| \geq p$ must have a rule (or chain of rules) such that indefinitely (or removed altogether).

3.7 Further Discussion of Context-Free Languages and Grammars

TODO: Not sure what we did here.

Chapter 4

Turing Machines

4.1 Pushdown Automata

A machine that has a stack and reads input from a tape. It can recognize a superset of Context-Free Languages.

A Pushdown Automata can be expressed as a 7-tuple:

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z, F \rangle$$

Where:

- Q is a finite set of states.
- Σ is the alphabet.
- Γ is the stack alphabet. It's the set of symbols that can go on the stack.
- $q_0 \in Q$ is the start state.

- $Z \in \Gamma$ is the initial stack symbol. It's the initial value in the stack.
- $F \subseteq Q$ is the set of accepting states.
- δ is the transition function that operates as follows:
  - Pop $\alpha \in \Gamma$ from the stack.
  - Read input $r \in (\Sigma \cup \{\varepsilon\})$.
  - See current state is $q \in Q$.
  - Move to a new state $q' \in Q$.
  - Push a new variable on the stack: $\alpha' \in \Gamma$.

  i.e. $\delta : \Gamma \times (\Sigma \cup \{\varepsilon\}) \times Q \to Q \times \Gamma^*$

4.2 The Turing machine model

Turing Machines are similar to PDAs but aren't exactly the same thing. Turing-machines consist of an infinitely long tape, a tape head, a state register and a function.

More formally, Turing Machines can be specified by a 7-Tuple:

$$M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$$

Where:

1. Q is a set of states

2. Γ is the tape alphabet. Basically what the tape can store.

3. $b \in \Gamma$ is the blank symbol. It's the "initial" value of most of the tape.

4. $\Sigma \in \Gamma$ is the set of input symbols.

5. $F \subseteq Q$ is the set of accepting states.

6. $\delta : Q \setminus F \times \Gamma \to Q \times \Gamma \times \{L, R\}$. δ is the transition function. It changes the state, writes to the tape, and moves the tape based on based on the value of the tape at the current location and starting state.

4.3 Variants of Turing Machines

TODO: Not sure what we did here.

4.4 Specifications of DTMs

TODO: Not sure what we did here.

4.5 The Church-Turing thesis

Basically, the thesis states that a function is algorithmically computable if and only if it is computable by a Turing machine.

4.6 Encoding schemes

We can encode anything as a set of 0's and 1's by serializing it. For example, we can denote an encoding of a DTM M as $\langle M \rangle$.

That's all I believe I need to write.

Chapter 5

Decidable and Recognizable Languages

5.1 Decidable and Turing-Recognizable Languages

Turing-Recognizable Languages are languages that can be "recognized" by a Turing Machine. i.e. A language P is said to be Turing Recognizable if there exists a Turing Machine M such that $L(M) = P$.

Decidable Languages are languages that can be "decided" by a Turing Machine. i.e. A language P is said to be decidable if there exists a Turing Machine M such that

for every input string $r \in \Sigma^*$, M will halt and output "accept" or "reject".

5.2 Closure Properties of Decidable and Turing-Recognizable Languages

Decidable Languages are closed under:

- Union
- Intersection
- Complementation
- Concatenation
- Kleene Star

Turing-Recognizable Languages are closed under:

- Union
- Intersection
- Concatenation
- Kleene Star

5.3 Some Decidable Languages

- $A_{DFA} = \{\langle B, w\rangle \,|\, B$is a DFA that accepts input string w
- $A_{NFA} = \{\langle B, w\rangle \,|\, B$is an NFA that accepts input string
- $A_{REX} = \{\langle B, w\rangle \,|\, B$is a regular expression that accep
- $E_{DFA} = \{\langle A\rangle \,|\, A$is a DFA and $L(A) = \emptyset\}$
- $EQ_{DFA} = \{\langle A, B\rangle \,|\, A$ and $B$are DFAs and $L(A) = L(B)\}$
- $A_{CFG} = \{\langle G, w\rangle \,|\, G$ is a CFG that generates $w\}$
- $A_{CFG} = \{\langle G, w\rangle \,|\, G$ is a CFG that generates $w\}$
- $E_{CFG} = \{\langle G\rangle \,|\, G$ is a CFG and $L(G) = \emptyset\}$
- $EQ_{CFG} = \{\langle G, H\rangle \,|\, G$ and $H$are DFAs and $L(G) = L(H)\}$
- Every context-free language is decidable.

5.4 A Non-Turing-Recognizable Languag

TODO: In this portion, I believe that we did the Halting Problem. Need to verify.

5.5 Undecidability of the Halting Problem

Description: Given a description of an arbitrary computer program, decide whether the program finishes running or continues forever. In other words, given a program and an input, please decide if the program eventually halts when run with that input? (or will it run forever)

Proof of undecidability: Suppose a solution to the halting problem H "accepts" input $H(P, I)$ if and only if P halts on I. Otherwise, it will always "reject".

Now we define the code Z as follows:

```
Program(String x):
    if Halt(x, x):
        Loop forever
    else:
        Halt
```

This is a pretty simple program, but if we run Z with input Z, we have impossible conclusions:

- In the case that $Halt(Z, Z)$ is true, we will loop forever. This is a contradiction.
- In the case that $Halt(Z, Z)$ is false, we will halt. This is a contradiction.

This function H cannot exist.

5.6 Reductions

One of the easiest ways to prove something is not decidable or not Turing-Recognizable is to establish a mapping from some other language you already know to be undecidable or not to be Turing-Recognizable.

We can define $\leq_m$ reductions (i.e. mapping reductions) as follows:

Being able to reduce problem A to B by using a mapping reducibility means that a computable function f exists that converts instances of A to B. If we have this f (called a reduction), we can solve A with a solver for B.

We want to find a way to "translate" every problem in A to a problem in B.

By finding $A \leq_m B$ and knowing some things about A, we know some things about B:

- $A \leq_m B$ and A is not decidable, then B is not decidable.
- $A \leq_m B$ and A is not Turing-Recognizable, then B is not Turing-Recognizable.

5.7 More Undecidable and Non-Turing-Recognizable Languages

TODO: Not sure what we did here.

5.8 Further Discussion of Turing Machine

TODO: Not sure what we did here.

Chapter 6

Computation Time

6.1 Time-Bounded Computation

TODO: Complete Me

6.2 The Time Hierarchy Theorem

I'm relatively sure it goes as follows:

$$P \subseteq NP \subseteq EXPTIME$$

TODO: Complete Me

6.3 Boolean Circuits and Their Relationship to Turing Machines

Any Turing Machine can be simulated by a boolean circuit, and every boolean circuit can be simulated by a Turing Machine.

6.4 P and EXP

P is the class of decision problems which can be solved by a DTM using a polynomial amount of computation time. i.e. Polytime or P time.

EXP is the set of all decision problems solvable by a DTM in $O(2^{p(n)})$ time, where $p(n)$ is a polynomial function of n.

6.5 Nondeterministic Turing Machines

Nondeterministic Turing Machines (NDTMs) are defined as a 6-tuple:

$$M = (Q, \Sigma, \iota, \sqcup, A, \delta)$$

Where:

- Q is a finite set of states.
- Σ is a finite set of symbols (the tape alphabet).

- $\iota \in Q$ is the initial state.
- $\sqcup \in \Sigma$ is the blank symbol.
- $A \subseteq Q$ is the set of accepting states.
- δ is the transfer function $Q \setminus A \times \Sigma \to Q \times \Sigma \times \{L, R\}$

The difference between a NDTM (NTM?) and a DTM is that NDTMs inhabit a set of states instead of a single state. They accept the state if any of their current states are accepted.

6.6 NP

We say that NP is the set of problems that are defined as follows:

$$\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$$

Alternatively, we can define NP using DTMs. A language L is in NP if and only if there exist polynomials p and q and a DTM M such that: [We call M a TODO what do we call M?]

- For all x and y, the machine M runs in time $p(|x|)$ on input $\langle x, y \rangle$.
- For all x in L, there exists a string y of length $q(|x|)$, M accepts $\langle x, y \rangle$.

- For all x not in L, there exists a string y of length $q(|x|)$, M rejects $\langle x, y \rangle$.

We can say that a language L is NP-Complete if L is NP and every A in NP is polynomial time reducible to B.

6.7 Polynomial-Time Mapping Reductions

Language A is polynomial time mapping reducible (or simply polytime reducible) to B, written $A \leq_p B$ if a polytime computable function $f : \Sigma^* \to \Sigma^*$ exists such that for every w, $w \in A \leftrightarrow f(w) \in B$.

If $A \leq_p B$ and $B \in P$, then $A \in P$.

3SAT is polytime reducible to CLIQUE. This means that if CLIQUE is solvable in polynomial time, so is 3SAT.

Set of related notes:

- If B is NP-Complete and $B \in P$, then P = NP
- If B is NP-Complete and $B \leq_p C$ for C in NP, then C is NP-Complete.
- 3SAT, CLIQUE, SAT, Vertex Cover, HamPath, Subset-SUM are all NP-Complete.

6.8 The Cook-Levin Theorem

The Cook-Levin theorem states that boolean satisfiability is NP-Complete.

That is, any problem in NP can be reduced to boolean satisfiability in polynomial time by a deterministic Turing Machine to the problem of determining whether a Boolean formula is satisfiable.

6.9 Further Discussion of Computability and Complexity theory