Exam 3 Stuff

Thursday, May 04, 2023 9:35 AM

- * Questions in the slides
- Ch7 slide 31

Combining Turing Machines

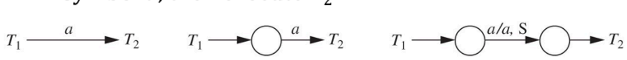
- Just as a large algorithm can be described as a number of sub-algorithms working in combination, we can combine several Turing machines into a larger composite TM.
- In the simplest case, if T_1 and T_2 are TMs, we can consider the composition T_1T_2: "first execute T_1, then execute T_2 on the result".
  - The set of states of T_1T_2 is the union of the sets of states of T_1 and T_2 (relabeled if necessary).
  - The initial state is the initial state of T_1.

Combining Turing Machines (cont'd.)

- The transitions of T_1T_2 include all of those of T_2 and all of those of T_1 that don't go to h_a.
- A transition in T_1 that goes to h_a is replaced by a similar transition that goes to the start state of T_2.
- It is important that the output of T_1 be a valid input configuration for T_2.
- We may use transition diagrams containing notations such as $T_1 \rightarrow T_2$, in order to avoid showing all the states explicitly.

Combining Turing Machines (cont'd.)

- We might use any of the above notations to mean "in state p, if the current symbol is a, then execute the TM T".
- Similarly we might use any of the following to mean "execute T_1, and if T_1 halts in h_a with current symbol a, then execute T_2".



- 2 Q abt TM
- 2 Q abt recursively enumerable langs
- Canonical vs lexicographical order
  - C – breadth first search

Enumerating a Language

- **Definition 8.8:** Let T be a k-tape Turing machine for some $k \geq 1$, and let $L \subseteq \Sigma^*$. We say T *enumerates* L if it operates such that the following conditions are satisfied:
  1. The tape head on the first tape never moves to the left, and no nonblank symbol printed on tape 1 is subsequently modified or erased;
  2. For every $x \in L$, there is some point during the operation of T

2. For every $x \in L$, there is some point during the operation of T when tape 1 has contents $x_1 \# x_2 \# \ldots \# x_n \# x \#$ for some $n \geq 0$, where the x_i's are also elements of L and $x_1, x_2, \ldots, x_n, x$ are distinct;
3. If L is finite, then nothing is printed after the # following the last element of L.

4/28/2023 Lecture 8 COSC 3302 60

Enumerating a Language (cont'd.)

- **Theorem 8.9:** For every language $L \subseteq \Sigma^*$,
  - L is recursively enumerable if and only if there is a TM enumerating L, and
  - L is recursive if and only if there is a TM that enumerates the strings in L in canonical order.
- **Reminder:** For the alphabet $\Sigma = \{a, b\}$, the *canonical order enumeration* of Σ^* is $\{\Lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$.
- **Note 1:** In the canonical order, the shorter strings precede longer strings and strings of the same length appear alphabetically.
- **Note 2:** Canonical order is different from the *lexicographic order* (or *strictly alphabetical order*), in which *aa* precedes *b*.
  - The lexicographic order enumeration of Σ^* is $\{\Lambda, a, aa, aaa, \ldots\}$, hence b and other strings will never be enumerated.

4/28/2023 Lecture 8 COSC 3302 61

- Context Sensitive Grammar
  - Monotone might not be on test

Context-Sensitive Languages and the Chomsky Hierarchy

- **Definition:** A context-sensitive grammar (CSG) is an unrestricted grammar in which no production is length-decreasing.
- In other words, every production is of the form $\alpha \rightarrow \beta$, where $|\beta| \geq |\alpha|$.
- **Note:** In fact, the above definition is for monotone grammars.
  - Harrison proved in 1969 that monotone grammars are equivalent to the context-sensitive grammars.
- A language is a context-sensitive language (CSL) if it can be generated by a CSG.
- CSGs cannot have Λ-productions, and CSLs cannot include Λ.
- We think of CSLs as a generalization of CFLs.

4/28/2023 Lecture 8 COSC 3302 80

The original definition of a context sensitive grammar [Chomsky; 1954]

- **Definition 1.** A formal grammar $G = (N, \Sigma, P, S)$, where N is a set of nonterminal symbols, Σ is a set of terminal symbols, P is a set of production rules, and S is the start symbol, is **context-sensitive** if all rules in P are of the form
  $\alpha A \beta \rightarrow \alpha \gamma \beta$
  where $A \in N$, $\alpha, \beta \in (N \cup \Sigma)^*$ and $\gamma \in (N \cup \Sigma)^+$.
- **Reminder.** A CFG production has the form $A \rightarrow \gamma$, so there is no context surrounding variable A.
- **Definition 2.** A formal grammar $G = (N, \Sigma, P, S)$ is in **standard form** if any rule of G has one of the following forms:
  1. $\alpha \rightarrow \beta$, where α and $\beta \in N^+$.
  2. $A \rightarrow a$, where $A \in N$ and $a \in \Sigma$.

4/28/2023 Lecture 8 COSC 3302 81

Weighted monotone grammars

- **Definition 3.** Given a formal grammar $G = (N, \Sigma, P, S)$, where N is a set of nonterminal symbols, Σ is a set of terminal symbols, P is a set production rules, and S is the start symbol, we define the weight of G denoted ψ(G), by $\psi(G) = \max\{|\beta|$, where $\alpha \to \beta \in P\}$. ■

- Recall that a grammar is monotone iff none of its productions is length-decreasing.
- **Theorem 2.** For any monotone grammar $G = (N, \Sigma, P, S)$, there exist a monotone grammar $G' = (N', \Sigma, P', S)$ of weight at most two such that $L(G) = L(G')$. ■
- **Proof.** Because G is monotone, $|\alpha| \leq |\beta|$ for any production $\alpha \to \beta$.
- If $|\alpha| \leq |\beta| \leq 2$, then the production $\alpha \to \beta$ is copied in G'.
- Let us consider $\alpha \to \beta$ for which $|\beta| > 2$. We'll proceed in a similar way as the equivalent Chomsky normal form.

4/28/2023 Lecture 8 COSC 3302 84

Weighted monotone grammars (cont'd)

- As stated in the previous slide, consider $\alpha \to \beta$ for which $|\beta| > 2$.
- Hence, $\alpha = x_1 x_2 \dots x_n$ and $\beta = y_1 y_2 \dots y_m$, where $n \leq m$ and $m > 2$, $x_1, x_2, \dots x_n, y_1, y_2, \dots y_m \in N$ (according to Theorem 1).
- We need m new additional non-terminal symbols, say, $z_1 z_2 \dots z_m$.
- We substitute the rule $x_1 x_2 \dots x_n \to y_1 y_2 \dots y_m$ the following set of weight at most two rules:
  - $x_1 \to z_1$
  - $z_1 x_2 \to y_1 z_2$
  - $z_2 x_3 \to y_2 z_3$
  - …
  - $z_{n-1} x_n \to y_{n-1} z_n$
  - $z_n \to y_n z_{n+1}$
  - $z_{n+1} \to y_{n+1} z_{n+2}$
  - …
  - $z_{m-1} \to y_{m-1} z_m$
  - $z_m \to y_m$ ■

4/28/2023 Lecture 8 COSC 3302 85

Example

- Let us continue with $G' = (\{A, B, X_a, X_b, X_c\}, \{a, b, c\}, A, P')$ with P':
  1. $X_a \to a$
  2. $X_b \to b$
  3. $X_c \to c$
  4. $A \to X_a X_b X_c$
  5. $A \to X_a A B$
  6. $X_b B \to X_b X_b X_c$
  7. $X_c B \to B X_c$
  - The rule $A \to X_a X_b X_c$ is transformed in: $A \to X_a Z_1, Z_1 \to X_b X_c$.
  - The rule $A \to X_a A B$ is transformed in: $A \to X_a Z_2, Z_2 \to A B$.
  - The rule $X_b B \to X_b X_b X_c$ is transf. in: $X_b B \to X_b Z_3, Z_3 \to X_b X_c$.
  - In fact, it is more efficient if the rule $X_b B \to X_b X_b X_c$ has $X_b B \to X_b Z_1$ instead. Why? (get a *)

4/28/2023 Lecture 8 COSC 3302 86

Monotone grammars are equivalent to the context-sensitive grammars

- **Theorem 3.** For any monotone grammar $G = (N, \Sigma, P, S)$, there exists a context-sensitive grammar $G' = (N', \Sigma, P', S)$ such that $L(G) = L(G')$. ■
- **Proof.** According to Theorems 1 and 2, there exists an equivalent grammar in standard form of weight at most two.
- Hence, we assume the rules of G have the form $\alpha \to \beta$ for which $|\alpha| \leq |\beta| \leq 2$. We distinguish six cases:
  1. $A \to a$ (sensitive with context (λ, λ))
  2. $A \to B$ (sensitive with context (λ, λ))
  3. $A \to B\,C$ (sensitive with context (λ, λ))
  4. $A\,B \to A\,C$ (sensitive with context (A, λ))
  5. $A\,B \to C\,B$ (sensitive with context (λ, B))
  6. $A\,B \to C\,D$ is not sensitive at any context. Hence …

4/28/2023 Lecture 8 COSC 3302 87

- Ch 8 slide 31-32

The original definition of a context sensitive

grammar [Chomsky; 1954]

- **Theorem 1.** For any formal grammar $G = (N, \Sigma, P, S)$, there exists a grammar G' in standard form equivalent to G.
- **Proof.** Let us assume that $G = (N, \Sigma, P, S)$ is not in standard form (otherwise G' can be taken as G).
- For each symbol $a \in \Sigma$, we associate a new non-terminal symbol $X_a \notin N \cup \Sigma$.
- We denote $N' = N \cup \{X_a \mid a \in \Sigma\}$.
- We define the function $f: N \cup \Sigma \to N'$ such that
  1. $f(y) = y$, if $y \in N$.
  2. $f(y) = X_y$, if $y \in \Sigma$.
- Function $f()$ can be extended to words (if $u = u_1 u_2 \dots u_k$, then $f(u) = f(u_1) f(u_2) \dots f(u_k)$).
- Consider $G' = (N', \Sigma, P', S)$ where $P' = \{f(u) \to f(v) \mid u \to v \in P\} \cup \{X_y \to y \mid y \in T\}$.
- The proof $L(G) = L(G')$ is rather straightforward. ■

4/28/2023 Lecture 8 COSC 3302 82

Example

- Let $G = (\{A, B\}, \{a, b, c\}, A, P)$, where P is given by:
  - $A \to a\ b\ c$
  - $A \to a\ A\ B$
  - $b\ B \to b\ b\ c$
  - $c\ B \to B\ c$
- (get a star) What is $L(G)$?
- What is the equivalent grammar in standard form?
- $G' = (\{A, B, X_a, X_b, X_c\}, \{a, b, c\}, A, P')$ with P':
  - $X_a \to a$
  - $X_b \to b$
  - $X_c \to c$
  - $A \to X_a X_b X_c$
  - $A \to X_a A\ B$
  - $X_b\ B \to X_b X_b X_c$
  - $X_c\ B \to B\ X_c$

4/28/2023 Lecture 8 COSC 3302 83

Monotone grammars are equivalent to the context-sensitive grammars (cont'd)

6. $A\ B \to C\ D$ is not sensitive at any context. Hence we need to rewrite this rule into three context-sensitive rules as follows (We consider E a new non-terminal symbol – similar to milk-water variable exchange example):
   a. $A\ B \to E\ B$ (sensitive with context (λ, B))
   b. $E\ B \to E\ D$ (sensitive with context (E, λ))
   c. $E\ D \to C\ D$ (sensitive with context (λ, D))
- In conclusion, G' has only context-sensitive rules and it is equivalent to G. ■

- Ch 8 smth like slide 35
  - Star questions around slide 74

Not Every Language is Recursively Enumerable

- We will now consider languages over an alphabet Σ and TMs with input alphabet Σ.
  - We will show that there are more languages than TMs to accept them. It follows that there must be many languages not accepted by any TM.
- The first step is to explain how it makes sense to talk about one infinite set being larger than another.
- We'll formulate two definitions:
  - What it means for two sets to be the same size;
  - What it means for one to be larger than another.

4/28/2023 Lecture 8 COSC 3302 104

Not Every Language is Recursively Enumerable (cont'd.)

- For finite sets, this is easy, because we know how to say that one *number* is equal to or bigger than another.
- **Definition 8.23:** In general, two sets A and B are the same size if there is a bijection $f : A \to B$. A is larger than B if some subset of A is the same size as B but A itself is not.
- **Definition 8.24:** A set A is *countably infinite* if there is a bijection $f : \mathbb{N} \to A$, or a list $a_0, a_1, \ldots$ of elements of A such that every element of A appears exactly once in the list.
  - A is *countable* if A is either finite or countably infinite.

Not Every Language is Recursively Enumerable (cont'd.)

- **Theorem 8.25:** Every infinite set has a countably infinite subset, and every subset of a countable set is countable.
  - For proof, see book.
- Even though the set $\mathbb{N} \times \mathbb{N}$ seems much larger than $\mathbb{N}$, it is countable and therefore the same size as $\mathbb{N}$.
- We can see this by forming a two-dimensional array with all the ordered pairs (i, j) and starting a spiral path at (0, 0) that hits each ordered pair.
- This is a way of listing the elements of $\mathbb{N} \times \mathbb{N}$.

4/28/2023 Lecture 8 COSC 3302 106

- Post Correspondence Question
  - Domino

Post's Correspondence Problem

- Here is a simple case: Think of each of the five rectangles in figure (*a*) as a domino
  - You have an unlimited supply of each.
  - The above array should match the below array.
  - Can we arrange them (with duplicates allowed) so that the top row of symbols matches the bottom row?
  - In this example, the answer is yes, as we see in figure (*b*)
  - In general, this is Post's Correspondence Problem.

10	01	0	100	1
101	100	10	0	010

(*a*)

10	1	01	0	100	100	0	100
101	010	100	10	0	0	10	0

(*b*)

Post's Correspondence Problem (cont'd.)

- **Definition 9.14:**
  - An instance of Post's correspondence problem (PCP) is a set $\{(\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n)\}$ of pairs, where $n \geq 1$ and each α_i and each β_i is a nonnull string over an alphabet Σ. ■
  - **The decision problem:** Given an instance of this type, do there exist a positive integer k and sequence of integers $i_1, i_2, \ldots, i_k$ with each i_j satisfying $1 \leq i_j \leq n$, satisfying $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_k} = \beta_{i_1}\beta_{i_2}\ldots\beta_{i_k}$? ■
  - An instance of the *modified* PCP (MPCP) adds the requirement that i_1 be 1.

4/28/2023 Lecture 9 COSC 3302 159

Post's Correspondence Problem (cont'd.)

Post's Correspondence Problem (cont'd.)

- **Definition 9.14:** (cont'd.)
  - Instances of *PCP* and *MPCP* are called *correspondence systems* and *modified correspondence systems*.
  - For an instance of either type, if it is a yes-instance we will say that there is a *match* for the instance.
  - We show that $Accepts \leq MPCP \leq PCP$ and deduce that both problems are undecidable (and that neither is completely unrelated to TMs after all).

4/28/2023 Lecture 9 COSC 3302 160

- SAT
  - 3SAT or NP or smth

Reductions and the Halting Problem

- We can often solve problems by reducing them to other, simpler ones.
- In this section, we consider reducing one decision problem to another.
- The two crucial features in a reduction are:
  1. For every instance I of the problem we start with, we must be able to obtain an instance $F(I)$ of the second problem algorithmically.
  2. The answer to the second question for the instance $F(I)$ must be the same as the answer to the original question for I.

4/28/2023 Lecture 9 COSC 3302 137

Reductions and the Halting Problem (cont'd.)

- **Definition 9.6:**
  - Suppose P_1 and P_2 are decision problems:
    - We say P_1 is *reducible* to P_2 ($P_1 \leq P_2$) if there is an algorithm that finds, for an arbitrary instance I of P_1, an instance $F(I)$ of P_2 such that the two answers (the answer to P_1 for the instance I, and the answer to P_2 for the instance $F(I)$) are the same.
  - If L_1 and L_2 are languages over alphabets Σ_1 and Σ_2
    - We say L_1 is *reducible* to L_2 ($L_1 \leq L_2$) if there is a Turing-computable function $f: \Sigma_1{}^* \to \Sigma_2{}^*$ such that for every $x \in \Sigma_1{}^*$, $x \in L_1$ if and only if $f(x) \in L_2$.

4/28/2023 Lecture 9 COSC 3302 138

Example

- Here is the **(SAT)** problem description.
- **Input:** *V* is a set of propositional variables, *F* is a propositional formula over *V* defined in Conjunctive Normal Form (CNF);
- **Output:** Does *F* have a satisfying assignment?
- Here is the **(3SAT)** problem description.
- **Input:** *V* is a set of propositional variables, *F* is a propositional formula over *V* defined in 3CNF;
- **Output:** Does *F* have a satisfying assignment?
- **Theorem.** Prove that (SAT) ≤ (3SAT).

The (SAT) Problem

- … is one of the most central decision problems in Computer Science.
- A *propositional variable*, *A*, is a variable that can take only two truth values, T (true) or F (false). ■
- **Example:**
  - The truth value of the assertion 'The door of this room is closed.' can be either true or false. ■
- Propositional variables can be combined into disjunctions ('or'), conjunctions ('and'), or negations ('negation').

- Godel #

Gödel Numbering

- In 1930's, Kurt Gödel developed a method of "arithmetizing" a formal axiomatic system by assigning numbers to statements and formulas.
  - This allowed him to describe relationships between objects in the system using relationships between the corresponding numbers.
- This led to Gödel's Incompleteness Theorem:
  - Any formal system comprehensive enough to include the laws of arithmetic must, if it is consistent, contain true statements that cannot be proved within the system.
- Gödel numbering will be useful in this chapter also.

Gödel Numbering (cont'd.)

- **Definition 10.17:** For every $n \geq 1$ and every finite sequence $x_0, x_1, \dots, x_{n-1}$ of n natural numbers, the Gödel number of the sequence is the number $gn(x_0, x_1, \dots, x_{n-1}) = 2^{x_0}3^{x_1} \dots (PrNo(n-1))^{x_{n-1}}$ where $PrNo(i)$ is the i^{th} prime.
  - Every sequence of length n is uniquely determined by its Gödel number.
  - If two sequences of different length have the same Gödel number, then they must agree except for the number of trailing 0's. ■

Gödel Numbering (cont'd.)

- Every positive integer is the Gödel number of some sequence of integers.
- For every n, the Gödel numbering determines a function from $\mathbb{N}^n$ to $\mathbb{N}$.
  - We will be imprecise and use the name gn for any of these functions;
  - All of them are primitive recursive;

- All of them are primitive recursive;
- The function *Exponent*: $\mathbb{N}^2 \to \mathbb{N}$, defined by *Exponent*$(i, x)$ = the exponent of *PrNo*(i) in x's prime factorization or 0 if i is 0,

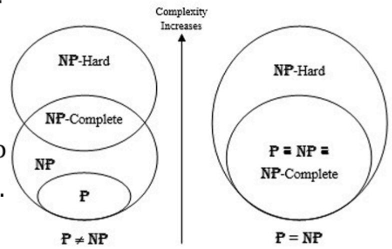
  is primitive recursive.
  - See book for details.

4/28/2023 Lecture 10 COSC 3302 194

- Ch 11 slide 27 diagram, also on 35?

Polynomial-Time Reductions and $\mathcal{NP}$-Completeness (cont'd.)

- **Definition 11.16:** A language L is $\mathcal{NP}$*-hard* if $L_1 \leq_p L$ for every $L_1 \in \mathcal{NP}$; L is $\mathcal{NP}$*-complete* if $L \in \mathcal{NP}$ and L is $\mathcal{NP}$-hard.
- $\mathcal{NP}$*-complete* is the intersection of $\mathcal{NP}$ and $\mathcal{NP}$-hard.
- We'll come back to this figure (slide 33).



4/28/2023 Lecture 11 COSC 5315 232