CS3063 Theory of Computing

Semester 4 (20 Intake), Feb – Jun 2023

Lecture 12

Turing Machines: Session 3

Sanath Jayasena

Announcements

- Assignment 2: was due 5th June

- Next week (the last week of the semester)
  - Lecture 13 and Lecture 14

- Please fill Student Feedback on Moodle

- Final Exam
  - 5th July, 9.00-11.00am

Outline:

Lecture 12

Turing Machines - 3

Turing Machines and Their Languages

- Recursive Languages & Recursively Enumerable Languages
- Unrestricted Grammars
- Context-Sensitive Grammars/ Languages
- Chomsky Hierarchy

PART 1

Outline:

Lecture 12

Turing Machines - 3

Turing Machines and Their Languages

- Recursive Languages & Recursively Enumerable Languages
- Unrestricted Grammars
- Context-Sensitive Grammars/ Languages
- Chomsky Hierarchy

Recall: Accepting vs Deciding

- A TM, T, with input alphabet Σ ***accepts*** a language L in Σ^* if L(T) = L
- A TM, T, ***decides*** L if T computes its characteristic function
  - That is: T decides L if T halts in state h_a for every string x in Σ^*, producing output 1 if x is in L and output 0 otherwise

- **Recognize**: use with care

Some Terminology

- Procedure
  - A finite sequence of instructions that can be mechanically carried out, given any input (for solving a problem)

- Algorithm
  - A procedure that terminates after a finite number of steps for any input

Some Terminology ...contd

- Recursive set
  - A set X for which we have an *algorithm* to determine whether a given element belongs to X or not
- Recursively-enumerable set
  - A set for which we have a *procedure* to determine belongingness

- A recursive set is recursively enumerable

Recursive Languages & Recursively Enumerable Languages

- A language L is *recursively enumerable* if there is a TM that accepts L
  - Also called ***Turing-acceptable***
- A language L is *recursive* if there is a TM that decides L
  - Also called ***Turing-decidable*** (or ***decidable***)

Recursive Languages & Recursively Enumerable Languages ...contd

- Properties
  - Every recursive language is recursively enumerable
  - i.e., there are recursively enumerable languages that are not recursive
  - If a TM accepts a language L, there can be strings not in L for which the TM loops forever (never produces output)

Recursive Languages & Recursively Enumerable Languages ...contd

- Properties
  - If L_1 and L_2 are recursively enumerable languages, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursively enumerable
  - If L is recursive, so is its complement $L' = \Sigma^* - L$

Enumerating a Language

- Enumerate: list all elements one at a time
  - Enumerable: can list all elements
- Definition: A TM Enumerating a Language
  - T is a k-tape TM and L is a subset of Σ^* ; we say T enumerates L if it operates as follows
    - Tape head on 1st tape never moves to left; no non-blank symbol on it never modified later
    - For every string x in L, at some point 1st tape will contain $x_1\#x_2\#...\#x_n\#x\#$ for some $n \geq 0$, where the strings $x_1, x_2, \ldots, x_n, x$ are distinct elements of L. If L is finite, nothing is printed after # following the last

Enumerating a Language

- From Definition
  - If L is finite
    - T can halt when all elements of L appear on 1st tape or continue moves without printing
  - If L is infinite
    - T will continue to move forever
- **Theorem 10.6, p. 369**
  - A language is recursively enumerable (i.e., can be accepted by some TM) **iff** it can be enumerated by some TM

Enumerating a Language …contd

- A language is recursive **iff** there is a TM that enumerates it in *canonical order*
  - Canonical order: 2 strings of different lengths, shorter one comes first; same length means alphabetical or numerical order

Enumerating a Language ...contd

- **Summary**
  - A language is ***recursively enumerable*** if there is an algorithm for listing its elements
  - A language is ***recursive*** if there is an algorithm for listing its elements in canonical order

PART 2

Outline:

Lecture 12

Turing Machines - 3

Turing Machines and Their Languages

- Recursive Languages & Recursively Enumerable Languages
- Unrestricted Grammars
- Context-Sensitive Grammars/ Languages
- Chomsky Hierarchy

Unrestricted Grammars

- Grammars, languages, abstract machines
  - Regular grammars, regular expressions, FA
  - CFG's, context-free languages, PDA

- A TM is the most general machine
  - More general grammar than CFG needed to generate a recursively-enumerable language

Unrestricted Grammars ...contd

- Recall: the “context-freeness” of CFG’s
  - LHS of a production has a single non-terminal and the *production can be applied whenever* that non-terminal appears in a string (*no matter what the context is*)
  - Allows to prove the pumping lemma for CFG’s
- Can relax the rules of CFGs
  - E.g., LHS of a production with >1 non-terminal

Unrestricted Grammars ...contd

- Example

$$\alpha A\beta \rightarrow \alpha\gamma\beta$$

  - Replace non-terminal A by γ (only when A is immediately preceded by α and followed by β; i.e., context dependent)
- Easier to write general productions as:

$$\alpha \rightarrow \beta$$

  - Production: simply a substitution of a string
  - But, LHS must contain ≥ 1 non-terminal

Unrestricted Grammars ...contd

- **Definition**: An *unrestricted* (or *phrase-structure*) grammar is a 4-tuple $G=(V, \Sigma, S, P)$ where:
  - V and Σ are disjoint sets of non-terminals and terminals, respectively
  - S is the start symbol
  - P is the set of productions of the form

$$\alpha \rightarrow \beta$$

    where α, β in $(V \cup \Sigma)^*$ and α contains at least one non-terminal

Example 1

- Unrestricted grammar for $L=\{a^i b^i c^i \mid i \geq 1\}$
  - (E.g. 10.1, p. 372)

$S \to FS_1$	$S_1 \to ABCS_1$	$S_1 \to ABC$
$BA \to AB$	$CA \to AC$	$CB \to BC$
$FA \to a$	$aA \to aa$	$aB \to ab$
$bB \to bb$	$bC \to bc$	$cC \to cc$

Example 1

- How to derive the string *aabbcc* from *L*?

S => FS_1 => FABCS_1 => FABCABC => FABACBC =>

FAABCBC => FAABBCC => *a*ABBCC => *aa*BBCC =>

*aab*BCC => *aabb*CC => *aabbc*C => *aabbcc*

Example 2

- Consider $L=\{ss \mid s \text{ is in } \{a,b\}^* \}$
  - (E.g. 10.2, p. 374)
  - Unrestricted grammar for L would be:

$S \rightarrow FM$	$F \rightarrow FaA$	$F \rightarrow FbB$
$Aa \rightarrow aA$	$Ab \rightarrow bA$	$Ba \rightarrow aB$
$Bb \rightarrow bB$	$AM \rightarrow Ma$	$BM \rightarrow Mb$
$F \rightarrow \Lambda$	$M \rightarrow \Lambda$	

Example 2

- How to derive the string $abbabb$ from L?

S => FM => FbBM => FbMb => FbBbMb => FbbBMb

=> FbbMbb => FaAbbMbb => FabAbMbb => FabbAMbb

=> FabbMabb => abbMabb => $abbabb$

Unrestricted Grammars & TMs

- Theorems
  - For any unrestricted grammar $G=(V, \Sigma, S, P)$, there is a TM, T, with input alphabet Σ and $L(T)=L(G)$
  - For any recursively enumerable language L, there is an unrestricted grammar G generating L

PART 3

Outline:

Lecture 12

Turing Machines - 3

Turing Machines and Their Languages

- Recursive Languages & Recursively Enumerable Languages
- Unrestricted Grammars
- Context-Sensitive Grammars/ Languages
- Chomsky Hierarchy

Context-Sensitive Grammars

- More general than CFG, less general than unrestricted grammars
- A *context-sensitive grammar* (CSG) is an unrestricted grammar in which every product has the form

  $\alpha \rightarrow \beta$ with $|\beta| \geq |\alpha|$

- A context-sensitive language (CSL) can be generated by a CSG

Context-Sensitive Grammars ...contd

- A language is *context-sensitive* iff it can be generated by a grammar in which every production has the form:

  $$\alpha A\beta \rightarrow \alpha X\beta$$

  where α, β and X are strings of non-terminals and/or terminals, $X \neq \Lambda$ and A is a non-terminal
- May allow A to be replaced by X depending on the context

Example

- CSG for $\{a^n b^n c^n \mid n \geq 1\}$
  - Example 10.5 on p. 381

$S \rightarrow \mathcal{A}BCS_1 \mid \mathcal{A}BC$

$S_1 \rightarrow ABCS_1 \mid ABC$

$BA \rightarrow AB$ $\quad CA \rightarrow AC$ $\quad CB \rightarrow BC$

$\mathcal{A} \rightarrow a$ $\quad aA \rightarrow aa$ $\quad aB \rightarrow ab$

$bB \rightarrow bb$ $\quad bC \rightarrow bc$ $\quad cC \rightarrow cc$

Linear-Bounded Automata (LBA)

- CSG correspond to linear-bounded automata that lie between PDA and TM
- An LBA is a non-deterministic TM with a limit on the length of tape
  - Head cannot move beyond specified bounds
  - Length moved bound linearly to input length
- (Ref: pp. 382-384 for Definition, details)

CSG/CSL and LBA

- Theorems
  - If L is a CSL, there is a LBA accepting L
  - If there is an LBA accepting the language L, a subset of Σ^*, then there is a CSG generating $L - \{\Lambda\}$

Chomsky Hierarchy

- We studied 4 classes of languages
  - Regular, context-free, context-sensitive and recursively-enumerable
- These are called the *Chomsky Hierarchy*
  - Chomsky denoted them as type 3, 2, 1 and 0
- Table next slide (p. 385)

The Chomsky Hierarchy

Type	Languages (Grammars)	Form of productions	Accepting Device
3	Regular	$A \to aB$, $A \to a$ (A, B in V, a in Σ)	Finite automaton
2	Context-free	$A \to \alpha$ (A in V, α in $(V \cup \Sigma)^*$)	Pushdown automaton
1	Context-sensitive	$\alpha \to \beta$ (α, β in $(V \cup \Sigma)^*$, $\lvert\beta\rvert \geq \lvert\alpha\rvert$, α has a V)	Linear-bounded automaton
0	Recursively enumerable	$\alpha \to \beta$ (α, β in $(V \cup \Sigma)^*$, α has a V)	Turing machine

unrestricted or *phrase-structure*

Languages not accepted by a TM?

- Not all languages are recursively enumerable
- Proof based on counting set elements
  - Main idea: the set of languages bigger than the set of TM's (a TM can accept one language)
  - Both are infinite sets but the 1st set is bigger !!

- Ref: Section 10.5 and Chapter 11
- More discussion next lecture

L12: Conclusion

- Today we discussed
  - Languages: recursive, recursively enumerable
  - Unrestricted grammars
  - Context-sensitive grammars
  - Chomsky Hierarchy