CS 133 : Automata Theory and Computability

Lecture Slides

Turing Machines and Decidability

Francis George C. Cabarle
fccabarle@up.edu.ph

Algorithms and Complexity Laboratory
Department of Computer Science
College of Engineering
University of the Philippines Diliman

Day 21

Decidability

Recursive Problems Concerning Regular Languages

Recursive Problems Concerning CFLs

- So far, we still ignore many real world limitations of computer programming languages.

- So far, we still ignore many real world limitations of computer programming languages.
- Some limitations: processing speed, size of the address space, data structures (not fundamental limits of computers)

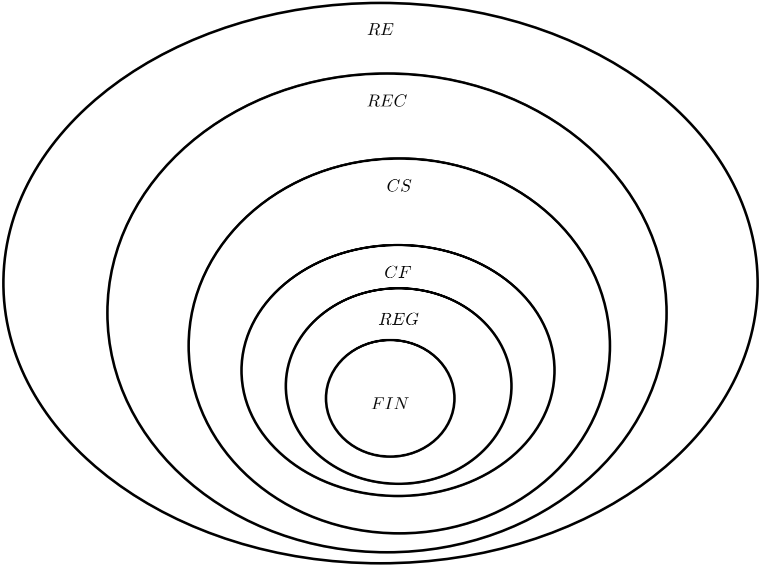
- So far, we still ignore many real world limitations of computer programming languages.
- Some limitations: processing speed, size of the address space, data structures (not fundamental limits of computers)
- As technology progresses we expect computers to increase in address-space size, main-memory size, etc

- So far, we still ignore many real world limitations of computer programming languages.
- Some limitations: processing speed, size of the address space, data structures (not fundamental limits of computers)
- As technology progresses we expect computers to increase in address-space size, main-memory size, etc
- Focusing on Turing machines, where many of these limitations do not apply, we can capture or reason about capabilities of some computing machine will be capable of doing, if not today, then at some time in the future.

- So far, we still ignore many real world limitations of computer programming languages.
- Some limitations: processing speed, size of the address space, data structures (not fundamental limits of computers)
- As technology progresses we expect computers to increase in address-space size, main-memory size, etc
- Focusing on Turing machines, where many of these limitations do not apply, we can capture or reason about capabilities of some computing machine will be capable of doing, if not today, then at some time in the future.
- We divide problems that can be solved by a TM into 2 classes:
  - **Class** *REC*: recursive or decidable languages, i.e. those that have an **algorithm** (i.e. a **TM** that **halts**, either accepting or rejecting input)
  - **Class** *RE*: recursively enumerable or recognizable languages, i.e. those that have an **algorithm** (i.e. TM) that **halt** if input is **accepted**, but **may loop forever** if input is **rejected**.

- So far, we still ignore many real world limitations of computer programming languages.
- Some limitations: processing speed, size of the address space, data structures (not fundamental limits of computers)
- As technology progresses we expect computers to increase in address-space size, main-memory size, etc
- Focusing on Turing machines, where many of these limitations do not apply, we can capture or reason about capabilities of some computing machine will be capable of doing, if not today, then at some time in the future.
- We divide problems that can be solved by a TM into 2 classes:
  - **Class** REC: recursive or decidable languages, i.e. those that have an **algorithm** (i.e. a **TM** that **halts**, either accepting or rejecting input)
  - **Class** RE: recursively enumerable or recognizable languages, i.e. those that have an **algorithm** (i.e. TM) that **halt** if input is **accepted**, but **may loop forever** if input is **rejected**.
- Relationship of REC and RE to closure properties?

Chomsky hierarchy so far...



Decidability

Recursive Problems Concerning Regular Languages

Recursive Problems Concerning CFLs

$$A_{DFA} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w\}$$

$$A_{DFA} = \{\langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w\}$$

Theorem

A_{DFA} is a recursive (decidable) language.

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$$

Theorem

A_{DFA} is a recursive (decidable) language.

$M =$ "On input $\langle B, w\rangle$, where B is a DFA and w is a string:

1. Simulate B on input w.
2. If the simulation ends in an accept state, accept. If it ends in a nonaccepting state, reject."

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

$$A_{NFA} = \{\langle B, w \rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Theorem

A_{NFA} is a recursive (decidable) language.

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Theorem

A_{NFA} is a recursive (decidable) language.

$N =$ "On input $\langle B, w\rangle$, where B is an NFA and w is a string:

1. Convert NFA B to an equivalent DFA C.
2. Run TM M on input $\langle C, w\rangle$.
3. If M accepts, accept; otherwise, reject."

$$A_{NFA} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts input string } w\}$$

Theorem

A_{NFA} is a recursive (decidable) language.

$N =$ "On input $\langle B, w\rangle$, where B is an NFA and w is a string:

1. Convert NFA B to an equivalent DFA C.
2. Run TM M on input $\langle C, w\rangle$.
3. If M accepts, accept; otherwise, reject."

Running TM M in stage 2 means incorporating M into the design of N as a subprocedure.

$$A_{REX} = \{\langle R, w \rangle \mid R \text{ is a regular expression that generates string } w\}$$

$A_{REX} = \{\langle R, w \rangle \mid R \text{ is a regular expression that generates string } w\}$

Theorem

A_{REX} is a recursive language.

$A_{REX} = \{\langle R, w \rangle \mid R \text{ is a regular expression that generates string } w\}$

Theorem

A_{REX} is a recursive language.

$M =$ "On input $\langle R, w \rangle$, where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent NFA A.
2. Run TM N on input $\langle A, w \rangle$.
3. If N accepts, accept; otherwise, reject."

$A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$

Theorem

A_{REX} is a recursive language.

$M =$ "On input $\langle R, w\rangle$, where R is a regular expression and w is a string:

1. Convert regular expression R to an equivalent NFA A.
2. Run TM N on input $\langle A, w\rangle$.
3. If N accepts, accept; otherwise, reject."

$\triangleright$ shows DFA, NFA or regular expression are equivalent because the machine is able to convert one form of encoding to another.

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Theorem

E_{DFA} is a recursive language.

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Theorem

E_{DFA} is a recursive language.

$T =$ "On input $\langle A \rangle$, where A is a DFA:

1. Mark the start state of A.
2. Repeat until no new states get marked:
   ▷ Mark any state that has a transition coming into it from any state that is already marked.
3. If no accept state is marked, accept; otherwise, reject."

$$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

Theorem

E_{DFA} is a recursive language.

$T =$ "On input $\langle A \rangle$, where A is a DFA:

1. Mark the start state of A.
2. Repeat until no new states get marked:
   ▷ Mark any state that has a transition coming into it from any state that is already marked.
3. If no accept state is marked, accept; otherwise, reject."

▷ Testing for emptiness.

$$EQ_{DFA} = \{\langle A, B\rangle \ \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

$$EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

Theorem

EQ_{DFA} is a recursive language.

$$EQ_{DFA} = \{\langle A, B\rangle \ \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$$

Theorem

EQ_{DFA} is a recursive language.

We construct a new DFA C from A and B, where C accepts only those strings that are accepted by either A or B but not by both.

$F =$ "On input $\langle A, B\rangle$, where A and B are DFAs :

1. Construct DFA C as described.
2. Run TM T on input $\langle C\rangle$
3. If T accepts, accept; otherwise, reject."

Decidability

Recursive Problems Concerning Regular Languages

Recursive Problems Concerning CFLs

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Theorem

E_{CFG} is a recursive language.

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Theorem

E_{CFG} is a recursive language.

$R =$ "On input $\langle G \rangle$, where G is a CFG:

1. Mark all terminal symbols in G.
2. Repeat until no new variables get marked:
   $\triangleright$ Mark any variable A where G has a rule $A \to U_1 U_2 \cdots U_k$ and each symbol $U_1, \cdots, U_k$ has already been marked.
3. If the start variable is not marked, accept; otherwise, reject."

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

Theorem

E_{CFG} is a recursive language.

$R =$ "On input $\langle G \rangle$, where G is a CFG:

1. Mark all terminal symbols in G.
2. Repeat until no new variables get marked:
   $\triangleright$ Mark any variable A where G has a rule $A \to U_1 U_2 \cdots U_k$ and each symbol $U_1, \cdots, U_k$ has already been marked.
3. If the start variable is not marked, accept; otherwise, reject."

$\triangleright$ Testing for emptiness.

$$A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates string } w\}$$

$$A_{CFG} = \{\langle G, w \rangle \ \mid \ G \text{ is a CFG that generates string } w\}$$

Theorem

A_{CFG} is a recursive language.

$$A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates string } w\}$$

Theorem

A_{CFG} is a recursive language.

Exercise: If G is in Chomsky normal form, any derivation of w has $2n - 1$ steps, where n is the length of w.

$$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$$

Theorem

A_{CFG} is a recursive language.

Exercise: If G is in Chomsky normal form, any derivation of w has $2n - 1$ steps, where n is the length of w.

$S =$ "On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2n - 1$ steps, where n is the length of w, except if $n = 0$, then instead list all derivations with 1 step.
3. If any of these derivations generate w, accept; if not, reject."

Theorem

Every context-free language is in REC.

Theorem

Every context-free language is in REC.

Let A be a CFL. We want to show that A is a recursive language.

Theorem

Every context-free language is in REC.

Let A be a CFL. We want to show that A is a recursive language.

Let G be a CFG for A and design a TM M_G that decides A.

$M_G =$ "On input w:

1. Run TM S on input $\langle G, w \rangle$.
2. If this machine accepts, accept; if it rejects, reject."

Theorem

Every context-free language is in REC.

Let A be a CFL. We want to show that A is a recursive language.

Let G be a CFG for A and design a TM M_G that decides A.

$M_G =$ "On input w:

1. Run TM S on input $\langle G, w \rangle$.
2. If this machine accepts, accept; if it rejects, reject."

▷ Testing or checking for **membership** in CFL is recursive (decidable).

Fin (wakas)

Thanks for the attention.

Questions?

Work on examples and exercises in:

[Sipser 2005] Chapter 4.1 , or

[Hopcroft et al 2001] Chapter 4.2, 7.3

References:

[Sipser 2005] M. Sipser. *Introduction to the Theory of Computation*: 2ed. PWS Publishing Company, 2005.

[Hopcroft, Ullman 1979] J. Hopcroft, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*: Addison-Wesley, 1979.

[Hopcroft et al 2001] J. Hopcroft, R. Motwani, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*: Addison-Wesley, 2001.

[Hernandez 2014] CS133 lecture slides of N.H.S. Hernandez, 2014