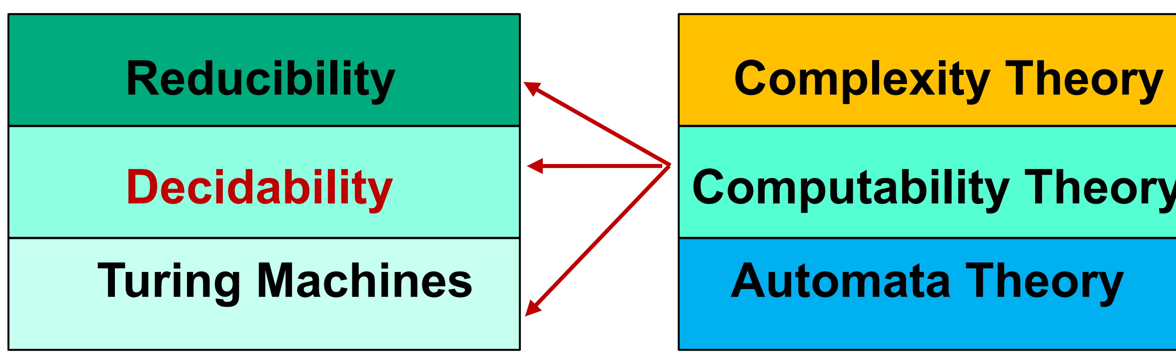
Decidability, part II



Decidable languages

- In last lecture, we looked at decidable problems concerning
  - Finite automata / regular languages
    - Acceptance ✓
      - DFA, NFA, Regular expression
    - Emptiness ✓
    - Equivalence ✓

- In this lecture, we look at decidability of problems concerning
  - Context-free grammars / languages
    - Generation
    - Emptiness
    - Equivalence

6) Context-free grammars: Generation

Let:

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

Theorem:

A_{CFG} is a decidable language

Proof: idea 1

- For CFG G and string w, we want to determine whether G generates w
- **Idea 1**: Use G to go through all derivations to determine whether any is a derivative of w
- Infinitely many derivations may have to be tried
- If G does not generate w, this algorithm will never halt
- Idea 1 gives a TM that is a *recognizer*, but not a *decider* for A_{CFG}
- Let us try another idea to turn it into a decider

Proof: idea 2

- Need to ensure that the algorithm relies on finitely many derivations
- We know that if the grammar G were in *Chomsky normal form*, any derivation of w has 2n-1 steps, where n is the length of w
- In that case, checking only derivations with 2n-1 steps to determine whether G generates w would be sufficient
- Only *finitely many* derivations exist
- We can convert G to CNF by using the procedure discussed in class on March 2 and March 4 (Example 2.10 in the book)
  - We also re-visited the procedure in HW5 and Mid Term 2

CNF: Quick Review

- Recall: Conversion to CNF can be performed via an algorithm that involves a series of find+replace steps. A TM should be able to perform these replacements:
  - Create new start state
  - Remove ε rules, add new rules where a variable could have been ε.
  - Eliminate all unit rules $A \rightarrow B$ by copying all rules for B into rules for A.
  - Create new variables for all terminals, replace all rules with more than just one terminal, with the new variable for that terminal. (e.g. $A \rightarrow cc$ becomes $A \rightarrow CC$ and $C \rightarrow c$)
  - Replace rules with more than 2 variables with 2-variable chains (e.g. $A \rightarrow BCD$ becomes $A \rightarrow BX$, $X \rightarrow CD$)
- New rules are more conducive to parsing grammar with an algorithm
  - A string of n terminals, generated by n variables, in turn generated by n/2 variables, and so on…
  - Total number of steps: $2n - 1$.

Proof: idea 2 (the complete proof)

The TM S for A_{CFG} is as follows:

$S =$ "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 one step.
3. If any of these derivations generate w, *accept*; if not, *reject*."

Side note:

- Determining whether a CFG generates a particular string is related to the problem of *compiling programming languages*
- The algorithm in TM S is inefficient; however, more efficient algorithms can be designed and are used in practice.

How about PDAs?

- Recall we have seen procedures for converting back and forth between CFGs and PDAs (reminders at the right)

- Hence, everything we say about decidability of problems concerning CFGs applies equally well to PDAs.

Thm 2:20: *A language is context free iff some pushdown automata recognizes it.*

7) Context-free grammars: Emptiness

Let:

E_{CFG} = {<G> | G is a CFG and L(G) = ∅}

Theorem:

E_{CFG} is a decidable language

Proof: Idea 1

- How about we use the TM S in the case we just considered (case 6)?

- The result states that we can test whether a CFG generates some particular string w
- To determine whether $L(G) = \emptyset$, the algorithm might try going through all possible w's, one by one
- But there are *infinitely many* w's to try, so this method could end up running for ever

- We need a different approach

Proof: Idea 2

- To determine whether the language of a grammar is empty, we need to test whether the *start variable* can generate a string of *terminals*

- The algorithm we will devise will do so by solving a more general problem

- It will determine ***for each variable*** whether the variable is capable of generating a string of terminals

- When the algorithm has determined that a variable can generate some string of terminals, the algorithm keeps track of this information by placing a mark on that variable

Proof: Idea 2

- First, the algorithm marks all the terminal symbols in the grammar
- Then, it scans all the rules of the grammar
- If it ever finds a rule that permits some variable to be replaced by some string of symbols, all of which are already marked, the algorithm knows that this variable can be marked, too
- The algorithm continues in this way until it cannot mark any additional variables.
- The TM R that implements this algorithm is shown in next slide

Proof: Idea 2

$R =$ "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:
3. Mark any variable A where G has a rule $A \rightarrow U_1 U_2 \cdots U_k$ and each symbol $U_1, \ldots, U_k$ has already been marked.
4. If the start variable is not marked, *accept*; otherwise, *reject*."

8) Context-free grammars: Equivalence

Let:

$$EQ_{CFG} = \{\langle G,H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$$

Consideration:

- We saw an algorithm that decides the analogous language EQ_{DFA} for finite automata
- We used the decision procedure for E_{DFA} to prove that EQ_{DFA} is decidable
- Because E_{CFG} also is decidable, can we use a similar strategy to prove that EQ_{CFG} is decidable?
- This won't work however because the class of CFL is ***not*** closed under *complementation* or *intersection*.

Theorem:

$$EQ_{CFG} \text{ is not decidable}$$

Proof: we will learn how to prove undecidability next lecture

9) The class of context-free languages

Theorem

Every context-free language is decidable

Proof Idea:

Let A be a CFL. Our goal is to show that A is decidable.

Idea 1:

- Convert a PDA for A directly into a TM
- Note hard to do since simulating a stack with the TM's tape is easy
- The PDA maybe *nondeterministic*, but we can handle that by converting the TM into a nondeterministic one
- Yet there is difficulty with this approach: some branches of the PDA *may go on forever*, reading and writing the stack without ever halting
- The simulating TM would *not* be a decider
- We need another idea

9) The class of context-free languages

Theorem

Every context-free language is decidable

Proof Idea 2:

We prove the theorem with the TM S that we designed to decide A_{CFG}

PROOF Let G be a CFG for A and design a TM M_G that decides A. We build a copy of G into M_G. It works as follows.

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*."

Conclusion: relationship among classes of languages