Curtin
University of Technology

Theoretical Foundations of Computer Science 300

Lecture 6

Decidability

Curtin University of Technology

Outline

- Decidable languages
- The Halting problem
- Turing-Unrecognizable language

Unit Learning Outcomes

- Understand recognisability and decidability, use the construction & mapping reducibility techniques to prove a problem decidable or undecidable.

Assessment Criteria

- **Describe** the classes of languages with respect to Turing Machines, Push-Down Automata and Finite Automata
- **Classify** problems as Turing Recognizable but Undecidable, or as Turing-Unrecognisable.
- **Prove** from first principles that some problems are Undecidable.

Curtin University of Technology

DECIDABILITY

Decidable Languages

Example: RL, DFA, NFA

Other Decidable Languages

Curtin University of Technology

Decidability

- Investigate the power of algorithms to solve problems
  - explore the limits of algorithmic solvability
  - the unsolvability of certain problems may be surprising

- Why study unsolvability ?
  - to be able to simplify or alter the problem before finding a solution
  - to avoid wasting time and resources on a problem with no solution
  - provides an important perspective on computation

Decidable languages

- Languages decidable by algorithms
- Example: algorithm to test if a string is a member of a CFL
  - related to the problem of recognizing and compiling programs
- First look at problems concerning finite automata
  - that can be solved by algorithms

Decidable problems of regular languages

- Algorithms for testing whether
  - a finite automata accepts a string
  - language of a finite automaton is empty
  - two finite automata are equivalent
- Languages to represent various computational problems
  - Example: acceptance problem of DFAs
  - Testing whether a particular finite automaton accepts a given string
  - Expressed as a language A_{DFA} containing encodings of all DFAs and the strings they accept

Curtin University of Technology

Language A_{DFA}

- A_{DFA}= {<*B*, *w*>|*B* is a DFA that accepts input string *w*}
- Problem of testing whether a DFA *B* accepts an input *w*
  - same as the problem of testing whether < *B*, *w*> is a member of the language A_{DFA}
- Other problems can be formulated similarly
  - in terms of testing membership in a language

Curtin University of Technology

Decidability of A_{DFA}

- Theorem: A_{DFA} is a decidable language
  - Proof idea: Construct a TM M that decides A_{DFA}
  - M= “On input <B, w>, where B is a DFA and w is a string:
  - Simulate B on input w.
  - If the simulation ends in an accept state, *accept*. If it ends in a non-accepting state, *reject*.”

Decidability of A_{DFA}

- Implementation details of the proof
  - Represent *B* as a list of its five components, *Q*, Σ, δ, q_0, and *F*.
  - On receiving input, *M* checks if it represents a DFA and a string. If not, *M* rejects.
  - *M* keeps track of *B*'s current state and its position in the input *w* by writing it down on tape
  - States and position are updated according to the transition function.
  - At the end of processing *w*, *M* accepts if *B* is in an accepting state; if not *M* rejects.

Decidability of A_{NFA}

- $A_{NFA} = \{\langle B, w\rangle \mid B$ is a NFA that accepts input string $w\}$
- Theorem: A_{NFA} is a decidable language.
  - Proof: Present a TM N that decides A_{NFA}. Design N to operate like M, simulating an NFA instead of a DFA.
  - Alternative: Let N use M as a subroutine. N first converts the NFA in the input to a DFA before passing it on to M.

Curtin University of Technology

Other decidable problems

- Whether a regular expression generates a given string
  - A_{REX}= {<B, w>|B is a regular expression that generates string w}
- Whether a DFA accepts any string at all
  - E_{DFA}= {<A>|A is a DFA and L(A)=ϕ}
- Whether two DFAs accept the same language
  - EQ_{DFA}= {<A,B>|A and B are DFAs and L(A)= L(B)}
- Similar decidable problems for CFGs
  - Also, every CFL is decidable

Curtin University of Technology

INTRODUCE THE HALTING PROBLEM

An Un-decidable Language

The Halting problem

Universal TM

The Halting Problem

- Computers are limited in a fundamental way
  - Even some ordinary problems are unsolvable by computers
- The general software verification problem
  - Given a program and a precise specification of the program, verify that the program performs as specified
  - Both the program and the specification are mathematically precise objects

The Halting Problem

- Objectives of discussion:
  - To get a feel for the type of problems
  - To learn techniques for proving un-solvability
- An un-decidable problem:
  - $A_{TM} = \{<M, w> | M \text{ is a TM and } M \text{ accepts } w\}$
  - A_{TM} is Turing-recognizable
  - Recognizers are more powerful than deciders
  - Requiring the TM to halt on all inputs restricts the languages it can recognize

Universal TM

- U= “On input <M, w>, where M is a TM and w is a string:
  - Simulate M on input w.
  - If M ever enters its accept state, accept; if M ever enters its reject state, reject.”
- This machine loops on input <M, w>, if M loops on w.
  - That is why U does not decide ATM.
- U is called universal because it can simulate any other TM.

Curtin University of Technology

THE HALTING PROBLEM IS UN-DECIDABLE

The Theorem

The Decider H

The Constructed D

Contradiction

Undecidability of Halting Problem

- A_{TM}= {<M, w>|M is a TM and M accepts w}

- Theorem: A_{TM} is un-decidable

Assumed Decider

- Proof: Assume that A_{TM} is decidable and obtain a contradiction

- Let H be a decider for A_{TM}.
  - $H(<M,w>)$ = *accept* if M accepts w and *reject* if M rejects w.

- Now construct a new TM D with H as a subroutine

Curtin University of Technology

Turing Machine D

- *D* calls *H* to determine what *M* does when input to *M* is its own description <*M*> and *D* does the opposite
  - *D*(<*M*>) = *accept* if *M* does not accept <*M*> and *reject* if *M* accepts <*M*>.

- *D* = "On input <*M*>, where *M* is a TM:
  - 1. Run *H* on input <*M*, <*M*>>.
  - 2. Output the opposite of what *H* outputs."

Curtin University of Technology

Undecidability of Halting Problem

- What if D is run with its own description as input ?
  - D(<D>) = accept if D does not accept <D> and reject if D accepts <D>.

- Contradiction because no matter what D does, it is forced to do the opposite
  - neither D nor H can exist.

CLASSES OF LANGUAGES

Complement of Language

Status of A_{TM}

Class of Languages

Turing-Unrecognizable language

- A_{TM} is un-decidable, but Turing-recognizable
- If both a language and its complement are Turing-recognizable, then the language is decidable.
  - For any undecidable language, either it or its complement is not Turing-recognizable.
  - The complement of a language is the set of all strings that are not in the language
- A language is co-Turing-recognizable if it is the complement of a Turing-recognizable language

Turing-recognizable and Co-Turing-recognizable languages

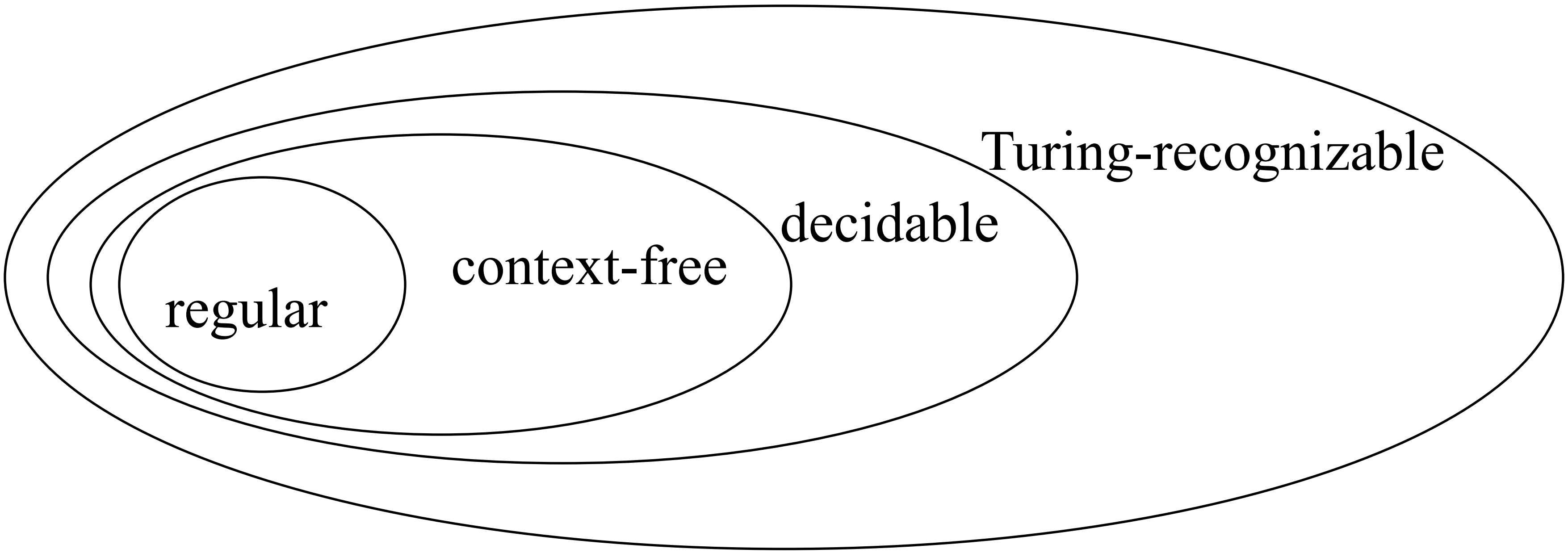
- If A is a decidable language, then both A and its complement are Turing-recognizable
- The complement of a decidable language also is decidable
- Let M_1 be the recognizer for A and M_2 be the recognizer for the complement of A.
- A TM that decides A:
  - "On input w:
  - 1. Run both M_1 and M_2 on input w in parallel.
  - 2. If M_1 accepts, accept; if M_2 accepts, reject."
- M has two tapes, one for simulating M_1 and another for M_2.
- M takes turns simulating one step of each machine

Curtin University of Technology

Corollary

- Complement of A_{TM} is not Turing-recognizable
- Proof:
  - ATM is Turing-recognizable.
  - If the complement of A_{TM} were Turing-recognizable, A_{TM} would be decidable
  - Since A_{TM} is not decidable, its complement is not Turing-recognizable

Relationship among classes of languages



Summary

- Decidable languages
- The Halting problem
- Turing-Unrecognizable language

Curtin University of Technology