### Review

一へ湯

2.2

(CAS)

「「「「「「「「「」」」」

ALC: NO

#### Overview

- Theory of computation: central areas:
   Automata, Computability, Complexity
- **Computability**: Is the problem solvable?
  - solvable and unsolvable
- **Complexity**: Is the problem easy to solve?
  - easy ones and hard ones
- Both deal with formal models of computation: **Turing machines**, recursive functions, lambda calculus, and production systems

#### Overview

# **Automata theory**: study of abstract machines and problems they are able to solve.

- closely related to formal language theory as the automata are often classified by the class of formal languages they are able to recognize.
- An abstract machine, also called an abstract computer, is a theoretical model of a computer hardware or software system
- FSM, PDA, Turing machine

#### Formal language: A set of strings over a given alphabet.

- In contrast to natural language
- Often defined by formal grammar
- Regular, context-free, D, SD

### Approach

- Language recognition framework:
  - transform a problem into a decision problem (verification) if it is not
  - encode the inputs as strings and then define a language that contains exactly the set of inputs for which the desired answer is yes
  - the original problem now becomes a language recognition problem
- Now, all problems have the same look
- Machines (FSM, PDA, TM ... ) are abstract computational models. They are used to classify problems.

#### Grammars, Languages, and Machines



#### Path

• Given input w, a maximal sequence of moves that M takes from the starting configuration

- A path P ends when it enters an accepting configuration, or there is no transition defined for its current configuration, i.e., it has no where to go.
- If P ends in an accepting configuration, P accepts w.
- If P ends in a non-accepting configuration, P rejects w.
- If there is some path of M that accepts w, M halts and accepts w.
- If all paths of M reject w, M halts and rejects w.
- It is possible that M neither accepts nor rejects w. In which case, none of its paths accepts w and some path does not reject w.
- Applies to all kinds of automata in the scope of our study.

#### **Different from the text**

### In Particular

- If M is deterministic, there's only **one** path.
- For DFSM and DPDA, the path is finite and guaranteed to end. Upon halting, it either accepts or rejects. So, DFSM and DPDA are guaranteed to halt (in at most |w| moves).
- Since DFSM = regular languages, the above halting property of DFSM implies that regular languages are in D.
- For DTM, the path maybe infinite. So DTM may not halt for some w.
  - It happens when M is a semi-decider and not a decider, and for w that is not in L(M)
  - Not that DTM = SD

#### In Particular

- If M is nondeterministic, there're maybe multiple paths.
- For NDFSM and NDPDA, if without  $\epsilon$ -transitions, they are guaranteed to halt in at most |w| moves. Otherwise, no.
- For each NDFSM M, there is an equivalent DFSM M'
  - "equivalent" means L(M) = L(M')
- For each NDPDA M, there an equivalent NDPDA M' that halts.
- Since NDPDA = context-free languages, the above halting property of NDPDA implies that context-free languages are in D.
- For NDTM, no such properties, as DTM does not even always halt.

#### **Recognizer and Decider**

- L(M): language accepted by M.
  - For each  $w \in L(M)$ , M halts
- Recognizer: M is a recognizer (acceptor) of L if M halts on any  $w \in L$  and returns "yes"
  - They are semi-deciders
- Decider: M is a decider of L if M always halts, returns "yes" if  $w \in L$  and "no" if  $w \not\in L$

#### Languages, Machines, and Grammars



#### Language Summary



#### Reduction

 $P_{old} \leq P_{new}$ 

- means that  $P_{old}$  is reducible to  $P_{new}$
- reduction from  $L_{old}$  to  $L_{new}$
- if  $L_{\text{new}}$  can be done,  $L_{\text{old}}$  can be done
- 1. Known  $P_{old}$  is not in D, we can show  $P_{new}$  is not in D
- 2. Known  $P_{old}$  is not in SD, we can show  $P_{new}$  is not in SD
- For both, no need to care about the efficiency of the reduction.

Can also be used in complexity to show NP-hardness:

- 3. Known  $P_{old}$  is NP-hard, we can show  $P_{new}$  is NP-hard
- Need to care about the efficiency of the reduction (polynomial).
- **Mapping reduction**: the most straightforward way of reduction is to transform instances of  $L_{old}$  into instances of  $L_{new}$
- $L_{old}$  is mapping reducible to  $L_{new}$  ( $L_{old} \leq_M L_{new}$ ) iff there exists some computable function *f* such that:

$$\forall x \in \Sigma^* \ (x \in L_{old} \leftrightarrow f(x) \in L_{new})$$
<sup>12</sup>

## The Complexity Zoo

The attempt to characterize the decidable languages by their complexity:

http://qwiki.stanford.edu/wiki/Complexity\_Zoo

Notable ones:

P: solution found by deterministic TM (algorithm) in polynomial time

- tractable
- context-free (including regular) languages are in P
- NP: solution found by nondeterministic TM (algorithm) in polynomial time
  - solution can be verified by DTM in polynomial time
- NP-complete: as hard as any one in NP & in NP (hardest ones in NP)
  - no efficient algorithm is known
  - require non-trivial search, as in TSP
- NP-hard: as hard as any one in NP (not necessarily in NP)
  - L is NP-complete if it is in NP and it is NP-hard
  - Intractable = not in P. But since it's believed P ≠ NP, loosely intractable
     = NP hard = NP complete + not in NP
  - If it's proved that P = NP, then intractable = not in P = not in NP

#### P and NP

Greatest unsolved problem in theoretical computer science: Is P = NP? <u>The Millenium Prize</u>

Two possibilities:



If P = NP, any polynomially verifiable problems would be polynomially decidable

#### **NP-Completeness**

The class of NP-complete is important, many of its members, like TSP, have substantial practical significance.
Two possibilities:



#### Strategy for Proving NP-completeness of L<sub>new</sub>

- Show that  $L_{new}$  belongs to NP
  - Exhibit an NDTM to decide it in polynomial time
    - Or, equivalently,
  - Exhibit a DTM to verify it in polynomial time
  - This establishes an upper bound on the complexity of  $L_{new}$
- Show that  $L_{new}$  is NP-hard by finding another NP-hard language  $L_{old}$  such that

$$L_{old} \leq_{\mathsf{P}} L_{new}$$

– This establishes a lower bound on the complexity of  $L_{new}$ 

