Decideability

Sipser pages 193 - 214

Decideability

- A class of problems is decidable if every problem can be answered Yes or No.
- We often look at classes that ask questions about languages and automata.
- We generally use the notation <P> to describe an encoding of a problem P in some way as input to a Turing machine.
- Turing machines are a good mechanism to talk about decideability.
  - Why?
  - What characteristics to Turing machines have?

Problems about DFA's

- A_{DFA} Does a DFA (B) accept some string (w)?
- E_{DFA} Is the language accepted by some DFA (B) the empty language (the empty set of strings).
- EQ_{DFA} Do two DFAs (A and B) accept the same language.

A_{DFA}

- Does a DFA (B) accept some string (w)?

- A_{DFA} = { <B,w> | B is a DFA that accepts input string w }

- Note that A_{DFA} is a language problem itself.
- Consider <B,w> to be the input language
- And the solution a Turing Machine that halts on all such input in either the accept or reject state

Representations

- Recall <B,w> is meant to represent a DFA and some input string.
- How might we represent this as input to a TM?
- $B = (Q, \Sigma, \delta, q_0, F)$

…	Q	…	#	…	Σ	…	#	…	δ	…	#	…	q_0	…	#	…	F	…	#

Checking A_{DFA}

…	Q	…	#	…	Σ	…	#	…	δ	…	#	…	q_0	…	#	…	F	…	#

- Let M be a TM that does the following
- Does the input represent a legal DFA
  - If not then reject
- Simulate B on input
- When finishing processing w, if the simulation is in an accepting state, then the TM accepts, else the TM rejects.

A_{NFA}

- How might we show that an NFA (C) accepts a string w?
- We might use a similar approach, encoding the NFA and its input on a TM tape <C,w> and then simulating the NFA.
- There is another approach!
- Since every NFA has an equivalent DFA (the subset construction) lets use the TM (M) of the last section.

N an TM that decides A_{NFA}

- N is a TM on input <C,w> where C is an NFA and w is a string
  1. Convert C into a DFA (D) using subset construction
  2. Run M on <D,w>
  3. If M accepts, then N accects, if M rejects, then N rejects

This is an example of an important strategy called reduction

How might we decide A_{RegExp}

E_{DFA} is decidable

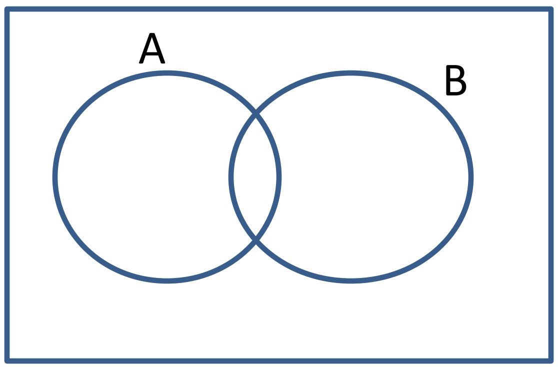
- T = On input <A> where A is a DFA
  - Mark the start state of A
  - Repeat until no new state is marked
    - Mark any state that has a transition coming into it from any state that is already marked
  - If no final state of A is marked, accept; other wise reject

EQ_{DFA} is decidable

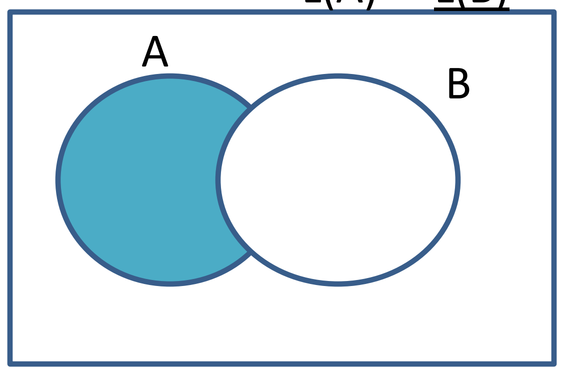
- To test if two DFAs decide the same language we will rely on several facts
  - DFA's are closed under intersection, union, and complement
  - EDFA is decidable (TM T from previous section)

Symmetric Difference

- $L(\ C\ ) = (\ L(A) \cap \underline{L(B)}\ ) \cup (\ \underline{L(A)} \cap L(B)\ )$

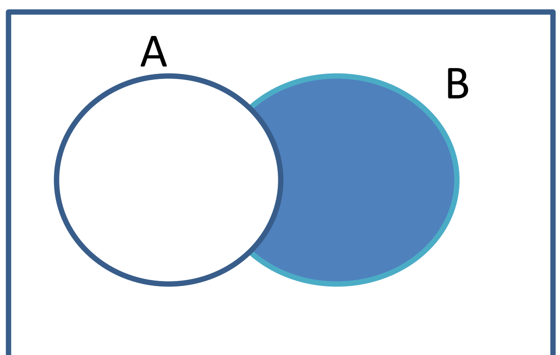


$L(A) \cap \underline{L(B)}$



If A and B are equal, then the symmetric difference is empty

$\underline{L(A)} \cap L(B)$



EQ_{DFA} is decidable

- F = On input <A,B> where A and B are DFAs
  1. Construct DFA C, the symmetric difference of A and B
  2. Run TM T (the one that decides E_{DFA}) on <C>
  3. If T accepts, then F accepts, if T rejects, then F rejects

Problems about CFG’s

- The following class of problems are discussed in the text. Be sure and read about them.
  – A_{CFG} Does a CFG (B) accept some string (w)?
  – E_{CFG} Is the language accepted by some CFG (B) the empty language (the empty set of strings).
    - This one is quite interesting, and not what one might expect. Pay close attention!
  – EQ_{CFG} Do two CFGs (A and B) accept the same language.

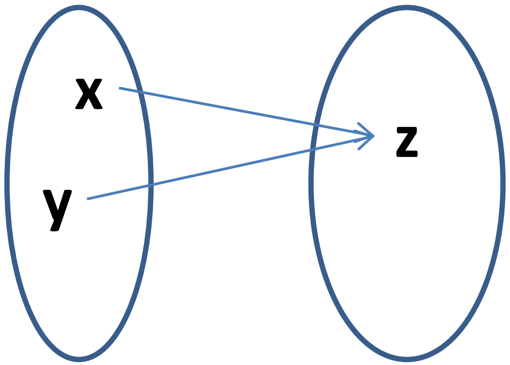
The size of infinite sets

- How can we tell if two sets have the same size?
- Easy for finite sets.
- Not so straightforward for infinite sets

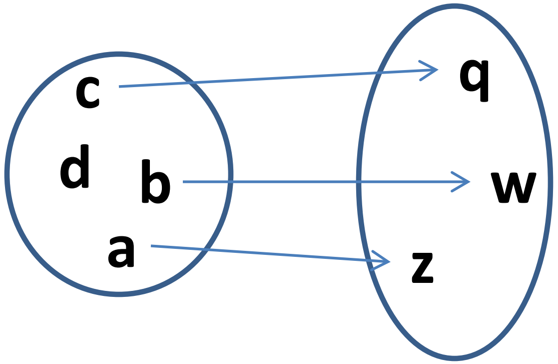
Two infinite sets have the same size if every element of one can be paired with the elements of the other

Properties of functions

In one-to-one functions this never happens



- One-to-one
  - A function, f, is one-to-one if it never maps two different elements of the domain to the same element of the range. $x \neq y \Rightarrow f(x) \neq f(y)$
- Onto
  - A function f is onto, if every element of the range is mapped to by some element of the domain



- Correspondence
  - A function is a correspondence if it is both one-to-one and onto

Naturals and the even-Naturals have the same size

n	f(n)
1	2
2	4
3	6
. . .	. . .

f(n) = n * 2

F is one-to-one, two numbers never map to the same element

F is onto, every even number is mapped to

Countable sets

- Definition
  - A set is countable, if it is finite, or if it is infinite, it is in correspondence to the Natural numbers

Rational numbers are countable

- Rational numbers, numbers exactly expressed as x/y, are countable.

1/1	1/2	1/3	1/4	1/5	1/6
2/1	2/2	2/3	2/4	2/5	
3/1	3/2	3/3	3/4		
4/1	4/2	4/3			
5/1	5/2				
6/1					

How can we establish a correspondance?

Can't travel along one row.

Or along one column

But along the diagonals

The real numbers are not countable

- We show no correspondence between R and N can exist.
- We use a classic argument (due to Cantor) called a diagonalization argument.

- First recall that every Real number can be expressed as an infinite decimal expansion. Example
  - 3.1415962…
  - 2.0000000…
  - 0.1250000…
  - 5.5555555…

Proof by contradiction.

- Assume that the Naturals and the Reals are in correspondence, then there exists a one-to-one, onto function, f : Nat -> Real

n	f(n)
1	3.14159…
2	55.5555…
3	0.12500…
4	0.50000…

A part of the coorespondence, f, between the naturals and the Reals

We show that f can’t be onto, thus it can’t be a correspondence, and hence the Reals can’t be countable

Consider the real between 0 and 1

n	f(n)
1	3.14159…
2	55.5555…
3	**0.12500…**
4	0.50000…

- All its digits are after the decimal point
- The nth digit after the decimal point is chosen different from the nth digit of the nth number, for example .2669…
- 2≠1
- 6≠5
- 6≠5
- 9≠0
- Note that no natural maps to this number. Suppose one did, let it be Z, but the Zth digit of f(Z) differs from our number in the Zth digit by construction.
- This is a contradiction, so our assumption that the Reals are countable must be false.

The set of all Turing Machines is countable

- Recall if Σ is finite, then Σ^* is countable
- We can write them all down
  - First all of length 0
  - Then all of length 1
  - Then all of length 2
  - Then all of length 3
- Each Turing Machine (M) has an encoding as <M> which is a string in Σ^*

The set of all infinite binary strings is not countable.

n	f(n)
1	1011001…
2	0010100…
3	1010111…
4	0110110…

- Diagonialization argument
- Consider 0101…
- Differs from the nth digit in the nth string

Characteristic functions of languages

- Consider the following function: F
- Given a finite alphabet Σ
- Given a language L over Σ
  - $L \subseteq \Sigma^*$
  - Σ^* is countable (thus so is L)
- F(i) = 1 if the ith string of Σ^* is in L, and 0 otherwise.
- We call F the characteristic function of L

The set of languages is not countable

- Given a finite alphabet Σ
- Consider the set of all languages, $\mathcal{L}$, over Σ^*
- Each language L in $\mathcal{L}$ has a characteristic function, F, which is an infinite sequence of 0's and 1's (I.e. an infinite binary sequence)
  - Eg consider L = { x | length of x is even }
  - F(ε)=1; F(0)=0; F(1)=0; F(11)=1; F(00)=1; F(01)=1; F(10)=1; …
- Thus, there is a correspondance between languages and infinite binary sequences.
- We know that the set of infinite binary sequences is not countable, so the set of languages over a finite alphabet Σ^*, can't be countable either!

There are languages not accepted by a Turing Machine. Sipser pg 178

- There are countable number of TMs
- A Turing Machine describes a language.
- There are uncountable number of languages.
- Thus some languages must not be describable by a TM.

The Halting problem

- Until now every problem we have looked at closely has been decidable.

- One might ask: "is any problem undecidable?"

- There is at least 1 undecidable problem A_{TM}
  - Acceptance by Turing Machine
  - Does an arbitrary TM accept an arbitrary input is undecidable

- This is an important result, both philosphically and computationally!

A_{TM} is Turing Recognizable!

- While not decidable, A_{TM} is Turing Recognizable.
- This depends upon the fact that there is a universal TM
- The universal Turing Machine takes <tm,input> and simulates “tm” on “input”.
- Note if “tm” does not halt on “input” neither does the universal TM halt on <tm,input>

R_{TM}, Recognizing a TM

- U = On input <M,w>, whem 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

- Note, if we had a way of determining that M would not halt on w, we could reject, but we don’t.

A_{TM} is undecidable

Sipser pg 179

- Proof by contradiction
- Assume that A_{TM} is decidable. By a TM called H
  - H(<M,w>) = accept if M accepts w, and reject if M does not accept w (I.e. M either rejects or loops)

- Then if M decides, we can make another machine D
  - D(<M>) = accept if H(<M>,<M>) rejects, and
    rejects if H(<M>,<M>) accepts

How a Turing machine M and H(<M>,w) are related.

M(w)	accept	reject	loop
H(<M,w>)	accept	reject	reject

How a Turing machine M and H(<M>,<M>) and D(<M>) are related.

M(<M>)	accept	reject	loop
H(<M>,<M>)	accept	reject	reject
D(<M>)	reject	accept	accept

The curious case when D is applied to itself.

D(<D>)	accept	reject	loop
H(<D>,<D>)	**accept**	**reject**	**reject**
D(<D>)	**reject**	**accept**	**accept**

Conclusion: A_{TM} is undecidable

D(<D>)	accept	reject	loop
H(<D>,<D>)	accept	reject	reject
D(<D>)	reject	accept	accept

- Since D(<D>) rejects if D(<D>) accepts we have reached a contradiction.
- So our original assumption that A_{TM} is decidable must be incorrect.
- Thus, A_{TM} is must be undecidable

Visualizing Diagonalization of A_{TM}

A table of the results of applying $H(<M_i><M_j>)$

H(I,j)	<M1>	<M2>	<M3>	<M4>	<M5>
M1	Accept	Reject	Reject	Accept	Reject
M2	Reject	Accept	Accept	Reject	Accept
M3	Reject	Reject	Reject	Reject	Reject
M4	Accept	Accept	Reject	Accept	Accept
M5	Reject	Accept	Reject	Accept	Reject

D is a TM so where is it in the Table?

H(<M>,<M>)	accept	reject	reject
D(<M>)	reject	accept	accept

H(I,j)	<M1>	…	<D>	…	<M5>
M1	Accept	Reject	Reject	Accept	Reject
…	Reject	Accept	Accept	Reject	Accept
D	Reject	Reject	?	Reject	Reject
…	Accept	Accept	Reject	Accept	Accept
M5	Reject	Accept	Reject	Accept	Reject

Definition

- A language is Turing co-recognizable if its complement is Turing recognizable.

- Recall the complement of a language is the language with all the strings not recognized by the original language.

M(w)	accept	reject	loop
CompM(w)	**reject**	**accept**	**accept**

Lemma

- A language, L, is decidable if and only if it is both Turing recognizable and Co-Turing recognizable.

- Two things to prove
  1. If L is decidable then it is both Turing and Co-Turing recognizable. This way is easy
  2. If L is Turing and Co-Turing recognizable, it is decidable

If M is Turing and Co-Turing recognizable, it is decidable

Turing recognizer	accept	reject	loop
$M_1(w)$	accept	reject	loop

Turing Co-recognizer	accept	reject	loop
$M_2(w)$	accept	reject	loop

- P(w) = run M_1(w) and M_2(w) in parallel
- If M_1 accepts, then P accepts.
- If M_2 accepts then P rejects.

P is a decider

- $P(w)$ = run $M_1(w)$ and $M_2(w)$ in parallel
- If M_1 accepts, then P accepts.
- If M_2 accepts then P rejects.

- Every string, w, is either in L (M_1 halts and accepts) or it is not (M_2 halts and rejects)
- So one of $M_1(w)$ or $M_2(w)$ must halt.
- P halts when either M_1 or M_2 halts, so P must Halt.
- So P is a decider that accepts all strings in L and rejects all strings not in L

Some languages aren't even recognizable!

Sipser pg 81

- Consider the language which is the complement of A_{TM} which we write $\underline{A_{TM}}$

- We prove that $\underline{A_{TM}}$ is not Turing recognizable using a proof by contradiction

- .

Proof

- Assume that $\underline{A_{TM}}$ is Turing recognizable
- We know A_{TM} is Turing recognizable
  - Sipser pg 174, theorem 4.11, Slide 29 in these notes
- Thus by our lemma A_{TM} is decidable
- We know that ATM is not decidable, which leads to a contradiction
- So our original assumption that $\underline{A_{TM}}$ is Turing recognizable must be flawed.

Review: Positive results

- Countable and uncountable Sets.
- Acceptance of Regular and Context Free languages is decidable.
- Equality of Regular and Context Free languages is decidable.
- Emptiness of Regular and Context Free languages is decidable.

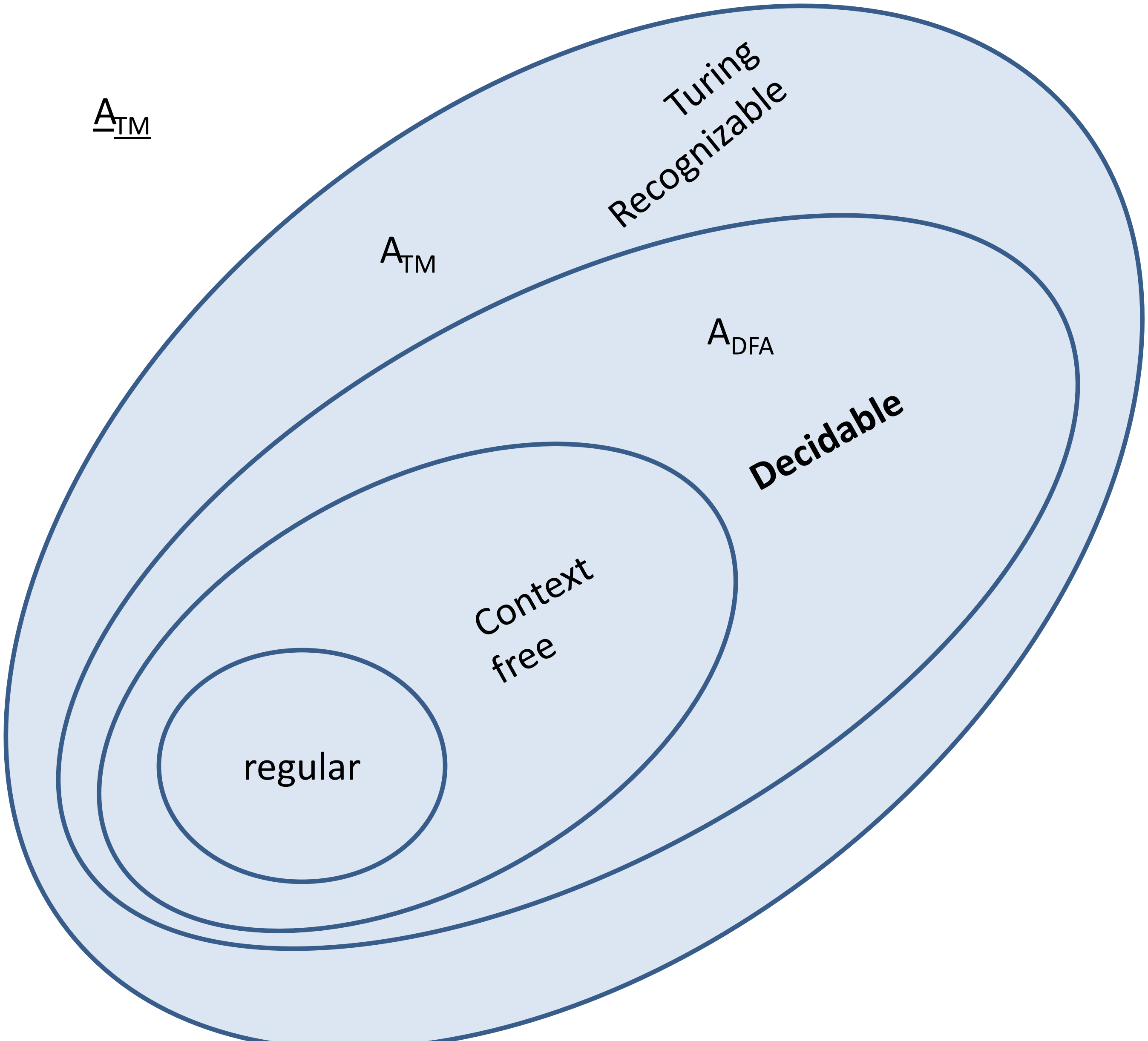
Review: Negative results

- There are uncountable Sets
  - The reals, infinite binary sequences, languages over a finite alphabet.

- There are languages not described by any Turing Machine.

- There is an un-decidable language
  - A_{TM} is undecidable
  - But, A_{TM} is Turing recognizable

- There is a language that is not even Turing recognizable! ($\overline{A_{TM}}$ the complement of A_{TM})


$\underline{A_{TM}}$
Turing
Recognizable
A_{TM}
A_{DFA}
Decidable
Context
free
regular