Reducability

Sipser, pages 187 - 214

Reduction

- Reduction encodes (transforms) one problem as a second problem.
- A solution to the second, can be transformed into a solution to the first.

- We expect both transformations (problem1 $\rightarrow$ problem2, and solution2 $\rightarrow$ solution1) to be computable.

Properties of Reduction

1. If A reduces to B, and B is decidable, then A must also be decidable, since a solution to B provides a solution to A.

2. If A reduces to B, and A is undecidable, then B must also be undecidable. If B were not undecidable, than we could use the solution to B to decide A (a contradiction since A is undecidable).

$HALT_{TM}$ is undecidable

- By reduction of A_{TM} to $HALT_{TM}$ we will show that $HALT_{TM}$ is undecidable

- $HALT_{TM}$ = { <M,w> | M is a TM and M halts on string w }

- Proof by contradiction
- Assume there exists a TM R that decides $HALT_{TM}$
- Construct TM S that decides A_{TM}
- We know A_{TM} is undecidable, so this is a contradiction
- Thus R cannot exist, so $HALT_{TM}$ is undecidable

The construction of S (using R)

- S =
  - On input <M,w>, run R on input <M,w>
  - If R rejects (M(w) does not halt) then reject
  - If R accepts (M(w) does halt) then
  - Simulate M on w until it halts.
  - If M has accepted then S accepts, else S rejects

Strategy

- This strategy for proving a language, L, undecidable
  - Reduce a known undecidable problem to a machine that decides L

- Is the preferred method for proving undecidability
- The most common target is A_{TM}
  - We proved A_{TM} undecidable by diagonalization.
- As we find new undecidable languages we have more targets for reduction.

E_{TM} is undecidable

- Testing if a TM only accepts the empty language is undecidable.
- Proof by reduction to A_{TM}

- Proof by contradiction
- Assume there exists a TM R that decides E_{TM}
- Construct a TM S that decides A_{TM}
- We know A_{TM} is undecidable, so this is a contradiction
- Thus R cannot exist, so E_{TM} is undecidable

S solves A_{TM}

- S(<M,w>) =
- Construct a modified version of M
  - $M_1(x)$ =
    - if x ≠w then reject.
    - If x=w, then simulate M on x, and accept if M does
    - M_1 accepts x only if x=w, and M accepts w.
    - *At most, M_1 accepts one string.*
- Run R on <M_1> (decide if M_1 accepts only the empty language).
- If R accepts, then S rejects
- if R rejects, then S accepts.
- if R rejects, then M_1 accepts some string β, but M_1 only accepts β if β =w and M accepts β, so M must accept β which must equal w).

Regular_{TM} is undecidable

- Testing if a Language recognized by TM can be recognized by a simpler language mechanism, regular expressions (or DFAs, NFAs, etc)
- Proof by reduction to A_{TM}

- Proof by contradiction
- Assume there exists a TM R that decides Regular_{TM}
- Construct a TM S that decides A_{TM}
- We know A_{TM} is undecidable, so this is a contradiction
- Thus R cannot exist, so Regular_{TM} is undecidable

We show how to reduce ATM to Regular$_{TM}$

- Construct S, that solves ATM, but relies on R to do so.
- S(<M,w>) = … R …
- We need a special Turing machine: H(x) which recognizes a regular language (Σ^*), if M accepts w, and recognizes an CF-Language (0^n1^n) if it rejects w.
  - H(x) =
    - if x has form 0^n1^n then H(x) accepts
    - If x does not have this form, if M(w) accepts then H(x) accepts, if M(w) rejects, then H(x) rejects

Use H to define S which decides A_{TM}

- S(<M,w>) = where M is a TM, w is a string
  - Run R on input <H>
  - If R accepts, then S accepts, if F rejects, then S rejects

- Recall that R decides if H recognizes a Regular language, but H recognizes Σ^* only if M decides w, Thus S decides A_{TM}, a known undecidable problem

EQ_{TM} is undecidable

- Testing if two Languages, both recognized by Turing Machines, both accept the same language.
- Proof by reduction to E_{TM}
  - All our other proofs gave been by reduction to A_{TM} , but this example lets us use another known undecidable language, E_{TM}, that decides if a language is the empty language.

- Proof by contradiction
- Assume there exists a TM R that decides EQ_{TM}
- Construct a TM S that decides E_{TM}
- We know E_{TM} is undecidable, so this is a contradiction
- Thus R cannot exist, so EQ_{TM} is undecidable

Construction of S

- Assume R decides EQ_{TM}, Construct S that decides E_{TM}.
- S(<M>) =
  - Run R on input <M,M_1>, where M_1 is a TM that rejects all inputs.
  - If R accepts, then S accepts, if R rejects, then S rejects

If R decides EQ_{TM} then S decides E_{TM}, which is known to be undecidable, a contradiction.

Computation History

- Recall a configuration (ID) has the form $\alpha\ q\ \beta$
  - where $\alpha, \beta \in \Gamma^*$ and $q \in Q$.
  - The string α represents the tape contents to the left of the head.
  - The string β represents the non-blank tape contents to the right of the head, including the currently scanned cell.
  - q represents the current state

- Recall configurations c_1,c_2 are related by
  - c_1 |- c_2
  - If the TM can legally move from c_1 to c_2

- A computation history (c_1, … , c_n) is a sequence of |- related configurations (each c_i |- c_{i+1})

Accepting (rejecting) Histories

- A computation history (c_1, … , c_n) is called an *accepting* history if c_1 is a start configuration and c_n is an accepting configuration

- A computation history (c_1, … , c_n) is called an *rejecting* history if c_1 is a start configuration and c_n is an rejecting configuration

If a TM does not halt on a given input, there does not exist an accepting (rejecting) history.

What about non-deterministic TMs?

Linear Bounded Automaton (LBA)

- An LBA is a restricted kind of TM
- Here the tape is restricted to the size of the input
- That is there is no infinite set of “Blank” symbols to the right of the input.

- We can stretch the amount of space available on the tape to a (constant * size of the input), by using extended alphabets.

LBA are quite powerful

- Language recognized by LBA include
  - A_{DFA}
  - A_{CFG}
  - E_{DFA}
  - E_{CFG}

- Surpisingly A_{LBA} is decidable
- { <M,w> | M is an LBA that accepts string w }

Lemma: Bound on number of configurations

- Let M be an LBA, with q states, and g Tape alphabet symbols, and a tape of size n
- There are exactly qng^n possible configurations

- Recall configuration has form α q β
- For a tape of size n, there are exactly n places where the we can place the q.
- There are g^n possible strings on the tape

A_{LBA} is decidable

- Let S be a TM that decides A_{LBA}. We construct S as follows.
- S(<M,w>) = *where M is a LBA, and w is a string*

- We must be careful, M might loop on w
- If its loops it must go through some configuration more than once.
- Keep a history of the configurations.
- Since there is a bounded number of configurations, call it B, any history longer than B must be looping

Constructing S

- S(<M,w>) = *where M is a LBA, and w is a string*
  - Simulate M on w for B steps or until it halts
  - If M has halted in an accepting state, S accepts
  - If M has halted in a rejecting state, S rejects
  - If M has not halted, it must be in loop, so S rejects

Key ideas

- A_{TM} is undecidable
- A_{LBA} is decidable
- Other problems on LBAs remain undecidable
- We use the configuration histories as a tool.

E_{LBA} is undecidable

- E_{LBA} decides if a LBA accepts the empty language
- Proof by contradiction
  - Assume E_{LBA} is decidable and then show A_{TM} must be decidable leading to a contradiction

- We use the familiar strategy:
  - A_{TM} (<M,w>) =
  - We create a particular LBA, B, that depends upon w, and use E_{LBA} to test B for emptiness.

Constructing B from M and w

- If M accepts w then there exists $(c_1, \ldots, c_n)$
  1. if c_1 is a start configuration and
  2. c_n is an accepting configuration
  3. Each consecutive pair c_i , c_{i+1} are related $c_i \mid\!- \; c_{i+1}$ by the transition function for M
- $B(<(c_1, \ldots, c_n)>) =$ accept if $(c_1, \ldots, c_n)$ is an accepting configuration history of M for w.
- Encoding $<(c_1, \ldots, c_n)>$ on the LBA tape
- $<c_1> \# <c_2> \# \; \ldots \; \# <c_n>$
- Encode each configuration, and separate by #

B uses M and w

- $B(<c_1> \# <c_2> \# \ \ldots \ \# <c_n>)=$
  1. Test if c_1 is a start configuration of M and w ($q_0 w_1 w_2 \ldots w_n$) AND
  2. c_n is an accepting configuration ($q_{accept}\,\alpha$)
  3. Each consecutive pair c_i , c_{i+1} are related $c_i \vdash c_{i+1}$ by the transition function for M

- A_{TM} (<M,w>) =
  – For a given M and w construct B as show above.
  – Use E_{LBA} to test B
    - if it accepts we know B accepts no strings, so no accepting history for w can exist, so A_{TM} should reject.
    - If it rejects we know there is at least one accepting configuration history for w, so A_{TM} should accept.

Using configuration histories

- We can use accepting and rejecting configuration histories do prove things about machines other than LBA

ALL_{CFG} is undeciadable

- ALL_{CFG} decides if a CFG accepts all strings.
- Proof by contradiction
  - Assume ALL_{CFG} is decidable and then show A_{TM} must be decidable leading to a contradiction
- We use the familiar strategy:
  - A_{TM} (<M,w>) =
  - We create a particular CFG that depends upon w

Strategy

- Create a CFG G that generates all strings iff M does not accept w.

- So if M does accept w, there must be some strings that G doesn't generate. We arrange for these strings to be strings of accepting computation histories for w under M. I.e. the CFG G in this case generates all strings that are not accepting computation histories.

-

The Turing machine

- A_{TM} (<M,w>) =
  - For the particular w, create a CFG, G, such that G does not generate the accepting computation configuration history for w, but generates all other strings of configurations.
  - Use ALL_{CFG} to test G
    - if it accepts we know G generates all strings, so there can be no accepting configuration for w, so A_{TM} should reject.
    - If it rejects we know there is at least one accepting configuration history, so A_{TM} should accept.

- A_{TM} is not decidable, so our assumption that ALL_{CFG} is decidable must be wrong.
- How do we define G?

Accepting configuration histories as languages

- $(c_1, \ldots, c_n)$ is called an *accepting* history
  1. if c_1 is a start configuration and
  2. c_n is an accepting configuration
  3. Each consecutive pair c_i , c_{i+1} are related $c_i \vdash c_{i+1}$ by the transition function for M

Such a sequence is a string, and the configurations that are accepting form a language (a set of strings) and a CFG could be designed to generate such a language.

Failure to be accepting

- $(c_1, \ldots, c_n)$ is called an *accepting* history
  1. if c_1 is a start configuration And
  2. c_n is an accepting configuration And
  3. Each consecutive pair c_i , c_{i+1} are related $c_i \mid\!- \; c_{i+1}$ by the transition function for M

- $(c_1, \ldots, c_n)$ fails to be accepting when
  1. c_1 is a *not* start configuration *OR*
  2. c_n is *not* an accepting configuration *OR*
  3. *some* consecutive pair c_i , c_{i+1} is *not* related $c_i \mid\!- \; c_{i+1}$ by the transition function for M

Design a PDA

- All CFG can be converted into PDA
- A PDA can be converted into a TM
- We don't care about how efficient the TM is, we are not going to run it. We are going to let the (non-existent) ALL_{CFG} TM analyze it.

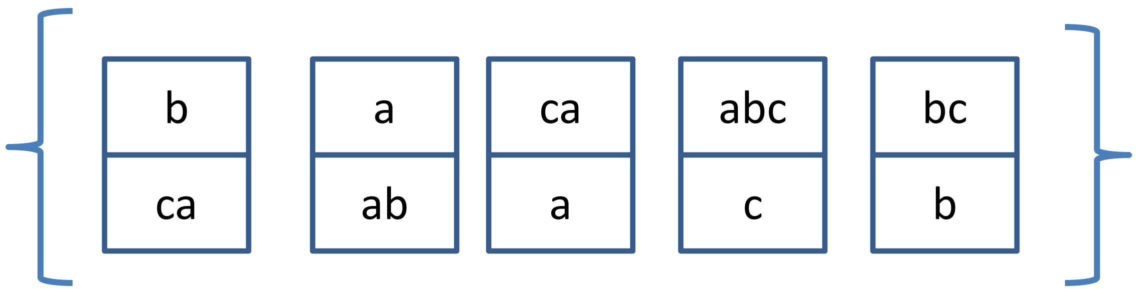
- The machine has three steps
  1. c_1 is a *not* start configuration *OR*
  2. c_n is *not* an accepting configuration *OR*
  3. *some* consecutive pair c_i , c_{i+1} is *not* related $c_i \mid\!- \, c_{i+1}$ by the transition function for M
- See the text for one strategy for the design of the TM emulating the PDA.

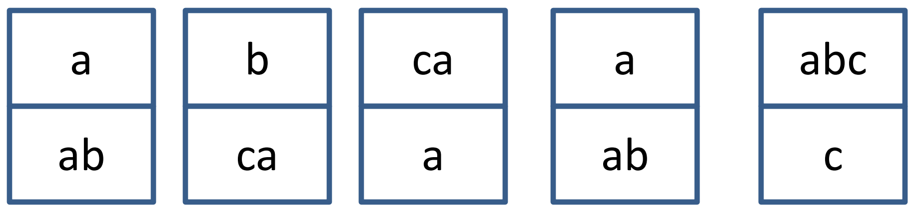
Post correspondence Problems

- The post correspondence problem looks for a solution to a simple game.
- Given a set of “dominos” like

b	a	ca	abc	bc
ca	ab	a	c	b

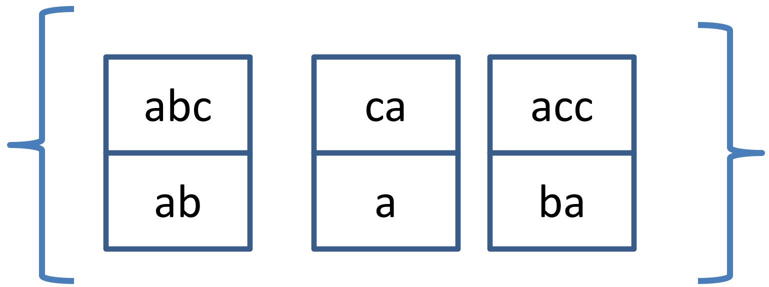
- Can one arrange the dominoes, side by side, such that the strings formed by concatenating top square and bottom square strings are the same.
- One can use each domino 0 or more times





abcaaabc
abcaaabc

For some set of dominos, no matches may be possible



Why are no matches possible here?

PCP = { <P> | P is an instance of the
Post correspondence problem with a match }

PCP is undecidable

PCP is undecidable

- Proof by contradiction
- Assume PCP is decidable
- Then build a TM for A_{TM} that uses PCP

- Given TM, M, and string, w, strategy depends upon finding an configuration accepting history for w. We show that such a history can be encoded as a PCP game
- I.e. we define a set of dominos, and if that set has a match, then the match would give an accepting configuration history for w, thus deciding A_{TM} which is known to be undecidable, leading to a contradiction.

The TM machine

- A_{TM} (<M,w>) =
  - Create a particular PCP game , p, from M and w such that a match for p is an accepting configuration history for w under M.
  - Use PCP to solve p
    - If p is solvable, then the match is an accepting configuration history for w, so A_{TM} accepts
    - If p is insolvable, then A_{TM} rejects

- How do we create p?

Technical details

1. We need M to never attempt to move its head of the left hand side of the tape.
   1. We can construct M’ with this property where M’ accepts the same strings as M.
2. If $w=\varepsilon$ we use a special symbol of the alphabet ␣ to represent ε.
3. We modify the PCP game to require that the match starts with the first domino in the set.

Constructing P

- Recall $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{accept},q_{reject})$
- We construct the dominos of P in seven parts.
- Each part place dominos that “simulate” some part of finding an accepting configuration history.

Step 1

1. Put

#
$\#q_0w_1w_2\ldots w_n$

As the first domino in the set. This forces the game to start with the initial configuration

Its clear we'll need more dominos that extend the top box of the domino if we are ever to find a match.

Step 2 – moving the head to the right

- For every a,b $\in \Gamma$, and
- every r,q $\in Q$ (where $q \neq q_{reject}$)
- If $\delta(q,a) = (r,b,R)$

qa
br

- Into the set of dominos

Step 3 – moving the head to the left

- For every a,b,c $\in \Gamma$, and
- every r,q $\in Q$ (where $q \neq q_{reject}$)
- If $\delta(q,a) = (r,b,L)$

cqa
rcb

- Into the set of dominos

Step 4 - cells not adjacent to the head

- For every a $\in \Gamma$
- put

a
a

- Into the set of dominoes

Step 5 – handling the markers (#)

- Put

#	#
#	␣#

- Into the set of dominos

Step 6 – catching up on accept

- For every $a \in \Gamma$
- Put $\left[\dfrac{a\ q_{accept}}{q_{accept}}\right]$ $\left[\dfrac{q_{accept}\ a}{q_{accept}}\right]$

- Into the set of dominos

Step 7 – cleaning up

- Add

$$\left[\frac{q_{accept}\ \#\#}{\#}\right]$$

- To the set of dominos

How does it work

- Each step towards acceptance supports only the addition of a single domino.
- Thus every accepting path leads to a match
- If there are no accepting paths, then the last cleanup steps are never possible so the top remains too short, and no match can be found.

Turing computable functions

- A function $\Sigma^* \rightarrow \Sigma^*$ is a computable function if some Turing Machine M, in every input w, halts with just f(w) on its tape.

- Some computable functions
  - Arithmetic functions like +, *, -, /, mod, etc.
  - Turing Machine description transformations
    - F(M) = M’ where M’ accepts the same strings a M but never tries to move its head of the left end of the tape

Mapping reducability

- A language A is mapping reducable to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w \in \Sigma^*$,

$$w \in A \Leftrightarrow F(w) \in B$$

- Mapping reducability creates a way to formally describe how to convert a question in A into a question in B

Unsurprising Theorems

Sipser page 208

1. $A \leq_m B$ and B is decidable then A is decidable

2. $A \leq_m B$ and A is undecidable, then B is undecidable

Old theorems in a new light

- $HALT_{TM}$
- Post correspondence
- E_{TM}

$HALT_{TM}$

- Find a computable function f such that
- $A_{TM} \leq_f HALT_{TM}$
- A_{TM} (<M,w>) = accept iff $HALT_{TM}$ (<M’,w’>) =accept
- Where f<M,w> = <M’,w’>

- f<M,w> =
  - create M’ <x> = run M on x
    - If M accepts then M’ accepts
    - If M rejects, enter a loop
  - F returns <M’,w>

A $\leq_m$ B and Turing Recognizability

- A $\leq_m$ B and B is Turing recognizable then A is Turing recognizable

- A $\leq_m$ B and A is not Turing recognizable then B is not Turing recognizable
  - Typically we let A be $\underline{A_{TM}}$ the complement of A_{TM}

Two ways to show not Turing recognizable

1. $\underline{A_{TM}} \leq_m B$ to show B is not Turing recognizable, by the second theorem on previous page

2. Because $A \leq_m B$ & $\underline{A} \leq_m \underline{B}$ mean the same
   1. Because of the definition of mapping reducability,

      F(A) = problem in B

      F($\underline{A}$) = problem in $\underline{B}$
   2. Thus can also use $A_{TM} \leq_m \underline{B}$ to show B is not Turing recognizable.

EQ_{TM} is neither Turing recognizable or co-Turing recognizable

- We must show two things
  1. EQ_{TM} is not Turing recognizable
  2. The complement of EQ_{TM}, $\overline{EQ_{TM}}$, is not Turing recognizable

Part 1: EQ_{TM} is not Turing recognizable

- Use the second method
- Show $A_{TM} \leq_m \overline{EQ_{TM}}$

- The reducing function F =
  - On input <M,w> construct the 2 TMs M_1 and M_2
    1. $M_1(x)$ on any input, x, reject
    2. $M_2(x)$ run M on w, if it accepts, accept
  - Output <M1,M2>

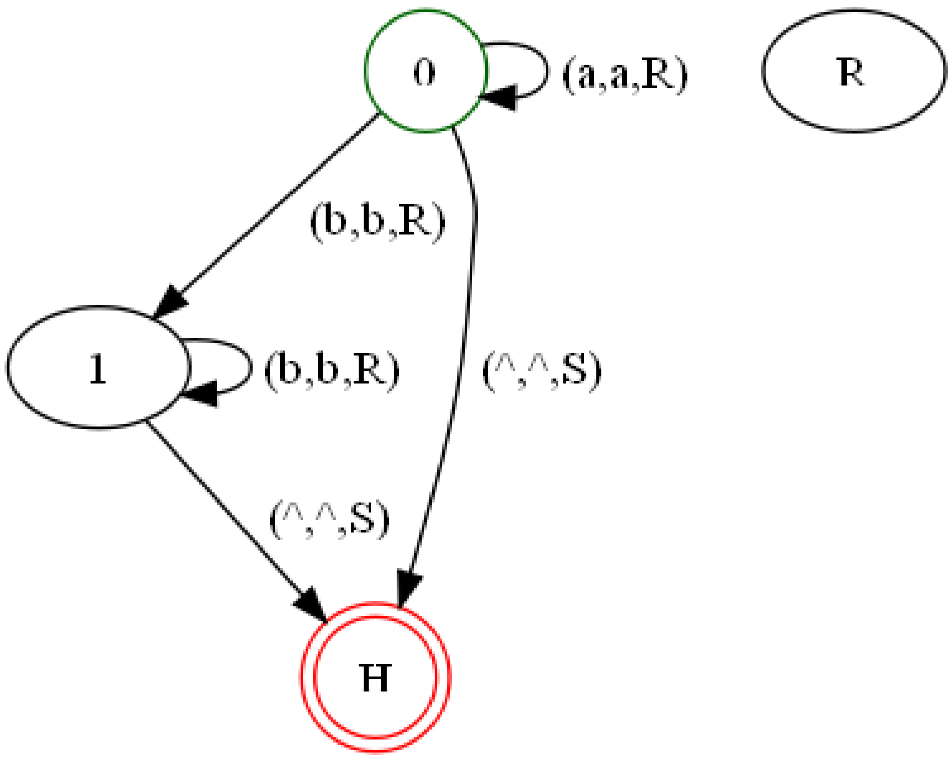
- Note M1 and M2 are equivalent only if M accepts w

Part 2: $\underline{EQ_{TM}}$ is not Turing recognizable

- Use the second method
- Show $\underline{A_{TM}} \leq_m \underline{EQ_{TM}}$ which is the same as
  $A_{TM} \leq_m EQ_{TM}$
- The reducing function G =
  - On input <M,w> construct the 2 TMs M_1 and M_2
    1. $M_1(x)$ on any input, x, accept
    2. $M_2(x)$ run M on w, if it accepts, accept
  - Output <M1,M2>
  - Note that M_1 and M_2 agree only if M accepts w

done

Example { $a^n b^m$ | $n,m \geq 0$} on the string "aab"



#	0a	0a	0b	0␣
#0aab	a0	a0	b1	␣1