Computability

Undecidability

- Informally, a problem is called unsolvable or undecidable if no algorithm exists that solves the problem.
- Algorithm
  - Implies a TM that decides a solution for the problem
- Decides
  - Implies will always give an answer

What about encoding TMs

- Consider the following
  - A_{TM} = { <M, w> | M is a TM and M accepts w}
    - This one we've seen – it's undecidable
    - However, The Universal Turing Machine U recognizes A_{TM}

The Halting Problem

- Consider the following
  - $HALT_{TM} = \{ <M, w> \mid M \text{ is a TM and } M \text{ halts on input } w\}$
  - Is the halting problem solvable?

The Halting Problem

- Suppose $HALT_{TM}$ is solvable?
  - Could that help us solve A_{TM} ? (which would be a contradiction)
  - Remember, to show A_{TM} is solvable, we need to be able to show that there exists a TM S that decides A_{TM} :
    - S = "On input <M, w> where M is a TM and w is a string:
      - Do these steps to show that S decides A_{TM}

The Halting Problem

- Suppose $HALT_{TM}$ is solvable – let TM R be a decider for $HALT_{TM}$.
  - Now consider this decider for A_{TM}:
    - S = "On input <M, w> where M is a TM and w is a string:
      - Run TM R on <M,w>
      - If R rejects (indicating it doesn't halt) then reject
      - If R accepts, then simulate M on w until it halts (which it must)
      - If M accepts, then accept. If M rejects, reject.
  - Thus S decides A_{TM}, which is a contradiction. It follows that $HALT_{TM}$ must be undecidable.

Observation

- Note that the proof that $HALT_{TM}$ is undecidable is much simpler than the proof that A_{TM} is undecidable
  - The strategy for proving that a problem Y is undecidable is to use an already-known-to-be-undecidable problem X
  - (We used a related approach a lot to show that languages were decidable as well)
  - Note – some choices of X will work much better than others

Reductions

- We're interested in ***reducing*** one problem to another

- Problem X reduces to problem Y

  - Is equivalent to saying

- If we have a solution to Y, that gives us a solution to X

Reductions

- For decidability:
  - We start off with Y known decidable, and
  - we show X reduces to Y
    - We conclude that X is decidable as well
  - Using this technique: we have a collection of known, decidable, languages to use as Y. Our task is to demonstrate the reduction of X to Y.

Reductions

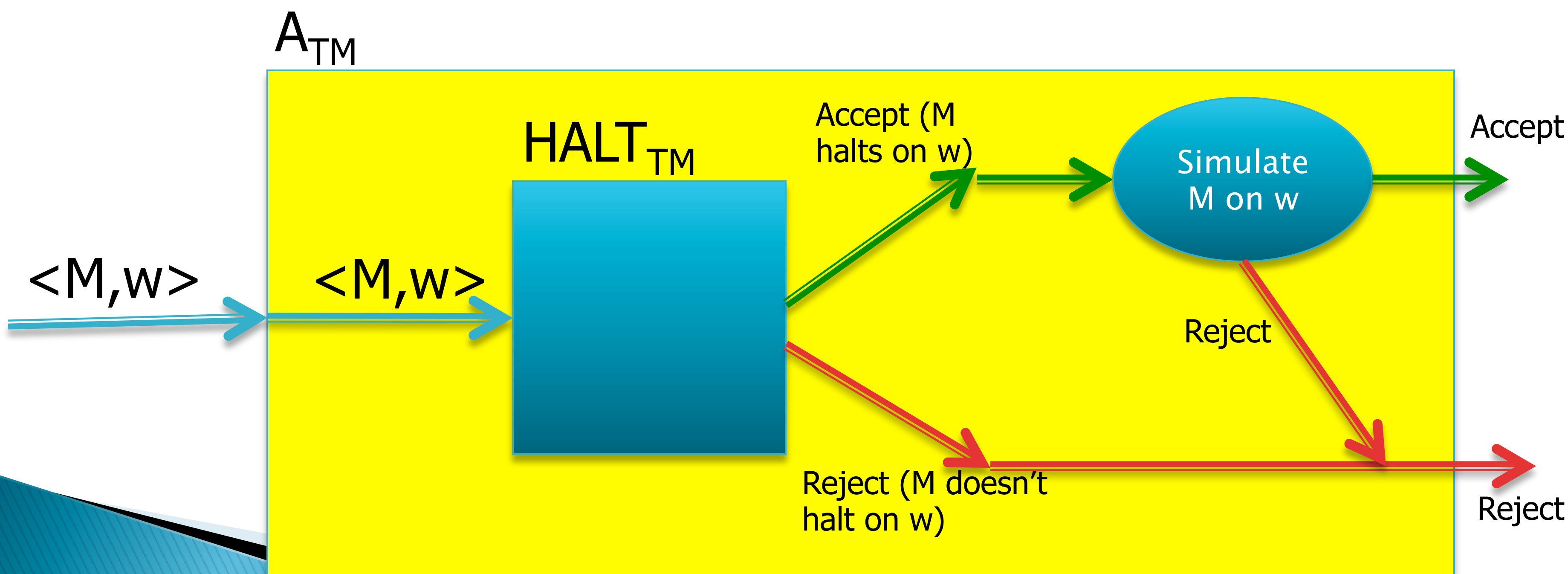
- For undecidability:
  - We have the contrapositive:
  - We start off with X known undecidable, and
  - we show X reduces to Y
    - We conclude that Y is undecidable as well
  - Using this technique: we have a collection of known, undecidable, languages to use as X. Our task is to demonstrate the reduction of X to Y.

Reductions

- For undecidability:
  - Undecidability follows by contradiction
    - We assume that Y is decidable
    - Once we show that X reduces to Y, that implies that X is decidable as well, which is a contradiction
    - Thus our assumption that Y is decidable must be false

Visualization of Undecidability Reduction

- Assumption: we have a black box that can decide what we're trying to show undecidable (Y – in this case $HALT_{TM}$).
- Having access to this black box allows us to solve another problem (the larger yellow box) that we already know isn't decidable (X – in this case A_{TM}). CONTRADICTION



Another Undecidability Example

- $E_{TM} = \{ <M> \mid M \text{ is a TM and } L(M) = \emptyset \}$
- We will build a contradiction by assuming that E_{TM} is decidable, and showing that this would imply A_{TM} is decidable.
- (Another way of thinking about it. We will show that the problem A_{TM} can be reduced to the problem E_{TM}. Since A_{TM} is known undecidable, E_{TM} must be undecidable as well.)

Another Example

- Recall A_{TM} = { <M, w> | M is a TM and M accepts w }
- Strategy: build a modified TM that will isolate w
- From <M,w> build a TM M_w that can be described as follows:
  - On input x:
    - If $x \neq w$ then reject.
    - Otherwise, run M on input w and if it accepts M_w accepts.
    - $L(M_w)$ can only have one possible string in it.

Another Example

- Note that
  - if M accepts w, then $L(M_w)$ is not empty.
  - If M rejects w (outright or by looping), then $L(M_w)$ is empty (since w is the only string M_w can possibly accept)

  - M accepts w if and only if $L(M_w)$ is not empty.

Another Example

- By our assumption, E_{TM} is decidable.
- $E_{TM} = \{ <M> \mid M \text{ is a TM and } L(M) = \emptyset \}$
  - There exists a TM, R that will decide it.
- Remember, to show A_{TM} is solvable, we need to be able to show that there exists a TM S that decides A_{TM} :
  - S = "On input <M, w> where M is a TM and w is a string:
    - Use the description of M and w to construct the TM M_w
    - Run R (our E_{TM} decider) on input $<M_w>$
    - If R accepts, reject. If R rejects, accept."
  - Thus S decides A_{TM}, which is a contradiction. It follows that E_{TM} must be undecidable.

One More Example: TM Equality

- $EQ_{TM} = \{<M_1, M_2> \mid M_1$ and M_2 are TMs, and $L(M_1) = L(M_2)\}$

- TM Equality is unsolvable.
  - Show this by contradiction. Assume that it is solvable (decidable) and prove that by using this assumption you can construct a TM that decides some other language we already know to be undecidable
  - Candidate languages: A_{TM}, $HALT_{TM}$, E_{TM}

One More Example: TM Equality

- For a given TM, M, compare M to a TM M_{empty} that accepts no strings.

- Note that
  - if the language of M is empty, then $L(M) = L(M_{empty})$.
  - If the language of M is not empty then $L(M) \neq L(M_{empty})$.

One More Example: TM Equality

- By our assumption, EQ_{TM} is decidable.
- $EQ_{TM} = \{<M_1, M_2> \mid M_1 \text{ and } M_2 \text{ are TMs}, L(M_1) = L(M_2)\}$
  - There exists a TM, R that will decide it.
- Remember, to show E_{TM} is solvable (and reach our contradiction), we need to be able to show that there exists a TM S that decides E_{TM} :
  - S = "On input $<M>$ where M is a TM:
    - Run R (our EQ_{TM} decider) on input $<M, M_{empty}>$ where M_{empty} is a TM that rejects all input
    - If R accepts, accept. If R rejects, reject.
  - Thus S decides E_{TM}, which is a contradiction. It follows that EQ_{TM} must be undecidable.

Rice's Theorem

- Rice's Theorem
  - Testing *any* non-trivial property of languages recognized by Turing machines is undecidable (see Problem 5.28)

Rice's Theorem

- Rice's Theorem
  - Formally, let P be a language consisting of Turing machine descriptors where P fulfills two conditions. First, P is nontrivial – it contains some, but not all TM descriptions. Second, P is a property of the TM's language – whenever $L(M_1) = L(M_2)$, we have $<M_1> \in P$ if and only if $<M_2> \in P$. Here M_1 and M_2 are any TMs. Then P is an undecidable language.

Rice’s Theorem

- Rice’s Theorem – what does it mean?
  - Formally, let P be a language consisting of Turing machine descriptors …
  - This means it has the form:
  - P = { <M> | M is a TM and … }

Rice's Theorem

- Rice's Theorem – what does it mean?
  - P is nontrivial – it contains some, but not all TM descriptions …
  - This means there must be at least one TM, M_{in}, whose description $<M_{in}>$ is in the language P, and at least one TM, M_{out}, whose description $<M_{out}>$ is not in the language P

Rice’s Theorem

- Rice’s Theorem – what does it mean?
  - P is a property of the TM’s language …
  - This means it has the form:
  - P = { <M> | M is a TM and L(M) … }

Applying Rice's Theorem

- Does Rice's Theorem imply that:

1. E_{TM} is undecidable?
2. A_{TM} is undecidable?
3. $FINITE_{TM}$ is undecidable? (for a TM, M, if L(M) is finite, then <M> is an element of $FINITE_{TM}$)
4. {<M> | M is a TM and L(M) is Turing-recognizable} is undecidable?
5. EQ_{TM} is undecidable?
6. {<M> | M is a TM and M has exactly four states} is undecidable?
7. $\{<M> \mid M$ is a TM and $\epsilon \in L(M)\}$ is undecidable?
8. $\{<M> \mid M$ is a TM and M halts on $\epsilon\}$ is undecidable?
9. $\{<M> \mid M$ is a TM and $|L(M)| \geq 0\}$ is undecidable?

Applying Rice's Theorem

- Does Rice's Theorem imply that:
  1. Yes
  2. No – not of proper form
  3. Yes
  4. No – trivial property of language; all TM satisfy
  5. No – not of proper form
  6. No – not of proper form (not property of language)
  7. Yes
  8. No – not of proper form (not property of language)
  9. No – trivial property of language; all TM satisfy

Post Correspondence Problem

- An undecidable problem about string matching.
- Valuable tool for proving other problems undecidable (CFG ambiguity)

- Given 2 lists of strings (each list with the same total number of strings; repeats allowed).
- Represent as a collection dominoes – each domino contains a string from list 1 and a corresponding string from list 2.

Post Correspondence Problem

- Can we pick a sequence of corresponding strings from the two lists (i.e. pick a collection of dominoes – repeats allowed) such that when we line them up, we get the same concatenated string on the top and the bottom?

Post Correspondence Problem

- Example:

List 1	10	01	0	100	1	0
List 2	101	100	10	0	010	00
	1	2	3	4	5	6

  - Choose a sequence of indices: 1,3,4
- List1: 10 0 100 List 2: 101 10 0

Post Correspondence Problem

- Is there a set of indices such that both lists produce the same string?

List 1	10	01	0	100	1	0
List 2	101	100	10	0	010	00
	1	2	3	4	5	6

- Try 1, 4, 6
- List 1: 101000 List 2 :101000

Post Correspondence Problem

- The Post Correspondence Problem (PCP) is undecidable (Thm. 5.15)

- Is PCP Turing-recognizable?

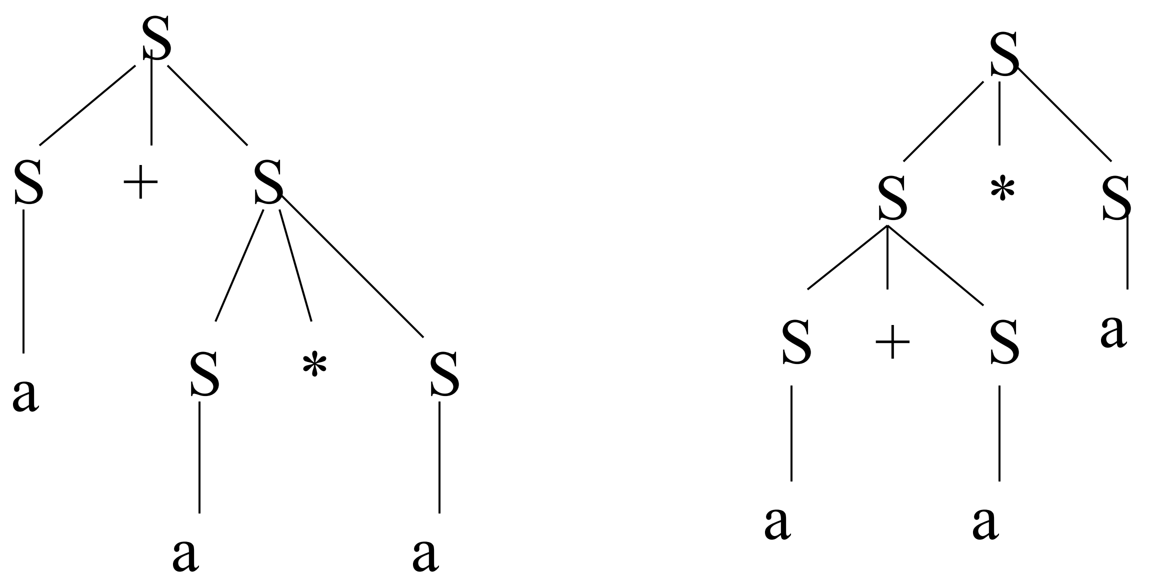
  - Yes, just list out all possible sequence of growing size
  - There are k strings to choose from, so only a finite number of possibilities for each size

  - This will discover all accept scenarios in finite time, but will certainly loop if it doesn't find anything

Ambiguity Revisited

- Ambiguity
  - Showing a particular grammar is ambiguous:
    - Find a string w in L(G) that has two derivations
  - Showing a particular grammar is <u>not</u> ambiguous is usually difficult.
  - <u>Making a statement about the ambiguity of any grammar is not possible.</u>

Recall: Ambiguity and Parse Trees



Same string: a + a * a, 2 derivations

CFG Ambiguity

- AMB_{CFG} = {<G> | G is a context free grammar and G is ambiguous}

- The CFG Ambiguity problem is unsolvable.

- Can be shown using the undecidability of PCP

- Assume AMB_{CFG} is solvable and arrive at a contradiction with PCP solvability

CFG Ambiguity

- Given an instance of PCP
  - 2 lists of strings T & B, all strings $\in \Sigma^*$
    - $T = (t_1, t_2, \ldots, t_k)$
    - $B = (b_1, b_2, \ldots, b_k)$

- Build a CFG, G with
  - Terminal set that includes Σ plus special symbols $\{ a_1, a_2, \ldots, a_k \}$ that are new terminals

CFG Ambiguity

- Instance of PCP
  - 2 lists of strings T & B, all strings $\in \Sigma^*$
    - $T = (t_1, t_2, \ldots, t_k)$
    - $B = (b_1, b_2, \ldots, b_k)$

- Productions of G
  - $S \rightarrow T \mid B$
  - $T \rightarrow t_1 T a_1 \mid t_2 T a_2 \mid \ldots \mid t_k T a_k$
  - $T \rightarrow t_1 a_1 \mid t_2 a_2 \mid \ldots \mid t_k a_k$
  - $B \rightarrow b_1 B a_1 \mid b_2 B a_2 \mid \ldots \mid b_k B a_k$
  - $B \rightarrow b_1 a_1 \mid b_2 a_2 \mid \ldots \mid b_k a_k$

CFG Ambiguity

- To show that deciding G lets us decide PCP, we need to show that PCP has a solution if and only if G is ambiguous
  - (then knowing whether G is ambiguous via the output of our assumed AMB_{CFG} decider will allow us to decide PCP – a contradiction)

CFG Ambiguity

- Assume G is ambiguous
  - A given string could have at most 1 derivation starting from T (similarly at most 1 derivation starting from B)
  - If a given string has 2 derivations, one must derive from T and the other from B
  - The string with 2 derivations will have the tail:
    - $a_{im} \ldots a_{i2}a_{i1}$ for some $m \geq 1$
    - On the T derivation the head will be $t_{i1}t_{i2}\ldots t_{im}$
    - On the B derivation the head will be $b_{i1}b_{i2}\ldots b_{im}$
    - $t_{i1}t_{i2}\ldots t_{im} = b_{i1}b_{i2}\ldots b_{im}$
    - (i1, i2, …im) is a solution to the PCP
- Conversely, if P has a match $t_{i1}t_{i2}\ldots t_{im} = b_{i1}b_{i2}\ldots b_{im}$
  - The string $t_{i1}t_{i2}\ldots t_{im}a_{im} \ldots a_{i2}a_{i1} = b_{i1}b_{i2}\ldots b_{im}a_{im} \ldots a_{i2}a_{i1}$ has a derivation from T and another from B
  - Hence the CFG is ambiguous

CFG Ambiguity

- Thus AMB_{CFG} decider allows us to decide PCP, which is a contradiction.
- So AMB_{CFG} is unsolvable.