CS 301 - Lecture 25
Computability and Decidability

Fall 2008

Review

- Languages and Grammars
  - Alphabets, strings, languages
- Regular Languages
  - Deterministic Finite and Nondeterministic Automata
  - Equivalence of NFA and DFA
  - Regular Expressions
  - Regular Grammars
  - Properties of Regular Languages
  - Languages that are not regular and the pumping lemma
- Context Free Languages
  - Context Free Grammars
  - Derivations: leftmost, rightmost and derivation trees
  - Parsing and ambiguity
  - Simplifications and Normal Forms
  - Nondeterministic Pushdown Automata
  - Pushdown Automata and Context Free Grammars
  - Deterministic Pushdown Automata
  - Pumping Lemma for context free grammars
  - Properties of Context Free Grammars
- Turing Machines
  - Definition, Accepting Languages, and Computing Functions
  - Combining Turing Machines and Turing's Thesis
  - Turing Machine Variations
  - Universal Turing Machine and Linear Bounded Automata
  - Recursive and Recursively Enumerable Languages, Unrestricted Grammars
  - Context Sensitve Grammars and the Chomsky Hierarchy
- Computational Limits and Complexity
  - Today: Computability and Decidability

Decidability

Consider problems with answer YES or NO

Examples:

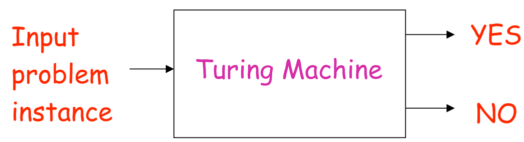
- Does Machine M have three states ?
- Is string w a binary number?
- Does DFA M accept any input?

A problem is decidable if some Turing machine decides (solves) the problem

Decidable problems:

- Does Machine M have three states ?
- Is string w a binary number?
- Does DFA M accept any input?

The Turing machine that decides (solves) a problem answers YES or NO for ***each input in the problem domain***



The domain is essential... part 1

Problem: is the following context-free language ambigous?

$$S \rightarrow abc$$

- Clearly we can decide this problem.

(the above grammar is not ambiguous)

The domain is essential... part 2

Problem: is an arbitrary context-free language ambigous?

- Clearly we can decide this problem this problem for ***some grammars*** in the domain.
- The problem is decidable only if we can answer this for ***all grammars*** in the domain

Some problems are undecidable:

which means:
there is no Turing Machine that
solves all instances of the problem

A simple undecidable problem:

The halting problem

The Halting Problem

Input: •Turing Machine M

•String w

Question: Does M halt on input w ?

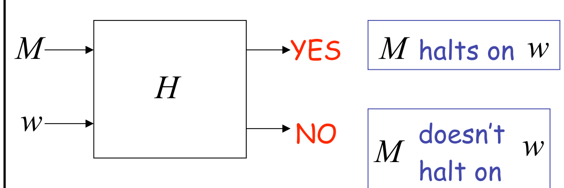
Theorem:

The halting problem is undecidable

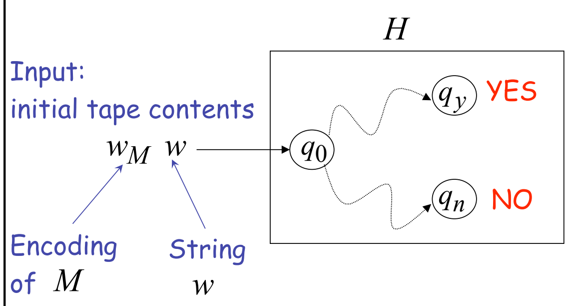
(there are M and w for which we cannot decide whether M halts on input w)

Proof: Assume for contradiction that the halting problem is decidable

Thus, there exists Turing Machine H that solves the halting problem



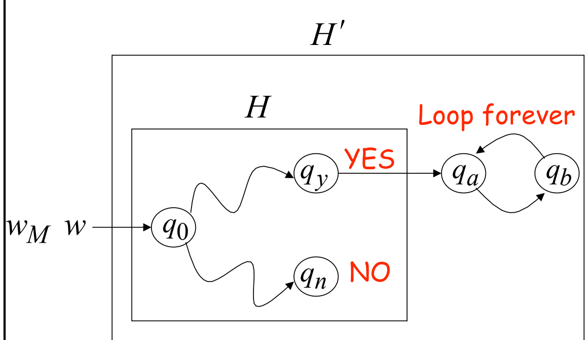
Construction of H



Construct machine H' :

If H returns YES then loop forever

If H returns NO then halt



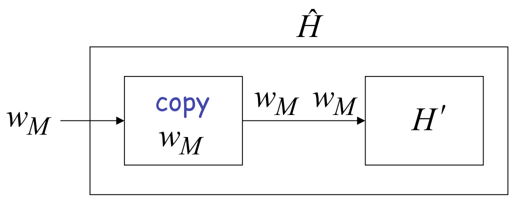
Construct machine $\hat{H}$:

Input: w_M (machine M)

If M halts on input w_M

Then loop forever

Else halt



Run machine $\hat{H}$ with input itself:

Input: $w_{\hat{H}}$ (machine $\hat{H}$)

If $\hat{H}$ halts on input $w_{\hat{H}}$

Then loop forever

Else halt

$\hat{H}$ on input $w_{\hat{H}}$:

If $\hat{H}$ halts then loops forever

If $\hat{H}$ doesn't halt then it halts

NONSENSE !!!!!

Therefore, we have contradiction

The halting problem is undecidable

END OF PROOF

Another proof of the same theorem:

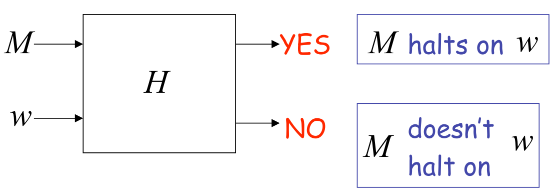
If the halting problem was decidable then every recursively enumerable language would be recursive

Theorem:

The halting problem is undecidable

Proof: Assume for contradiction that the halting problem is decidable

There exists Turing Machine H that solves the halting problem



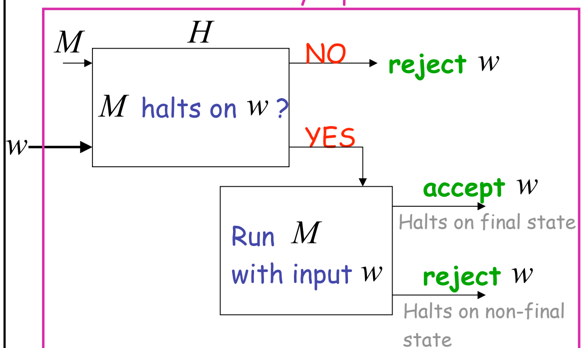
Let L be a recursively enumerable language

Let M be the Turing Machine that accepts L

We will prove that L is also recursive:

we will describe a Turing machine that accepts L and halts on any input

Turing Machine that accepts L
and halts on any input



Therefore L is recursive

Since L is chosen arbitrarily, every recursively enumerable language is also recursive

But there are recursively enumerable languages which are not recursive

Contradiction!!!!

Therefore, the halting problem is undecidable

END OF PROOF

The Membership Problem

Input:
- Turing Machine M
- String w

Question: Does M accept w ?

$$w \in L(M) ?$$

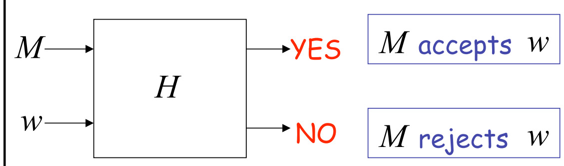
Theorem:

The membership problem is undecidable

(there are M and w for which we cannot decide whether $w \in L(M)$)

Proof: Assume for contradiction that the membership problem is decidable

Thus, there exists a Turing Machine H that solves the membership problem



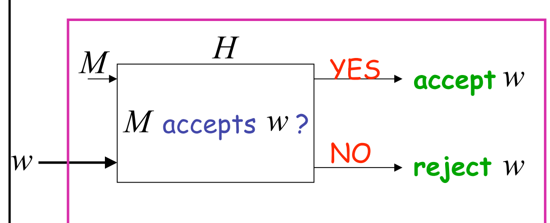
Let L be a recursively enumerable language

Let M be the Turing Machine that accepts L

We will prove that L is also recursive:

we will describe a Turing machine that accepts L and halts on any input

Turing Machine that accepts L and halts on any input



Therefore, L is recursive

Since L is chosen arbitrarily, every recursively enumerable language is also recursive

But there are recursively enumerable languages which are not recursive

Contradiction!!!!

Therefore, the membership problem is undecidable

END OF PROOF

Some Undecidable Problems

Halting Problem:

Does machine M halt on input w?

Membership problem:

Does machine M accept string w?

Are These Problems Undecidable?

State-entry Problem:

Does machine M enter state q on input w?

Blank-tape halting problem:

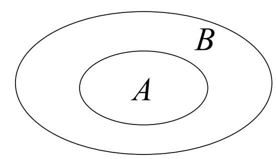
Does machine M halt when starting on blank tape?

Could start from scratch for each problem... instead could we use our previous results??

Reducibility

Problem A is reduced to problem B

If we can solve problem B then we can solve problem A



Problem A is reduced to problem B

If B is decidable then A is decidable

If A is undecidable then B is undecidable

Example: the halting problem

is reduced to

the state-entry problem

The state-entry problem

Inputs:
- Turing Machine M
- State q
- String w

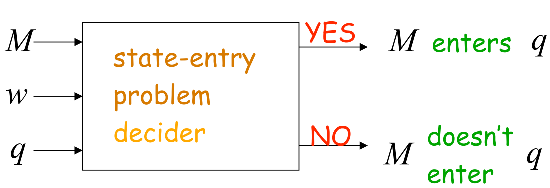
Question: Does M enter state q on input w ?

Theorem:

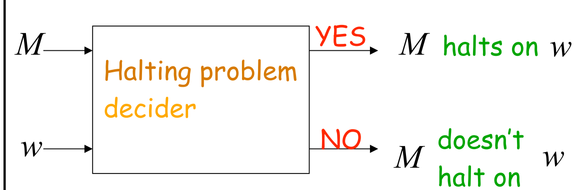
The state-entry problem is undecidable

Proof: Reduce the halting problem to the state-entry problem

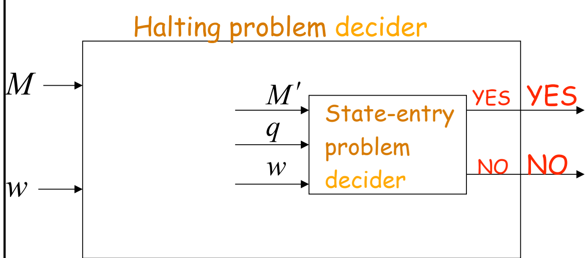
Suppose we have a Decider for the state-entry algorithm:



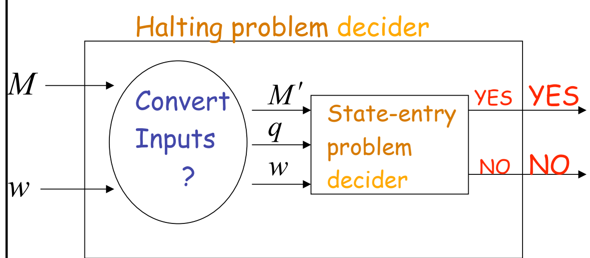
We want to build a decider for the halting problem:



We want to reduce the halting problem to the state-entry problem:

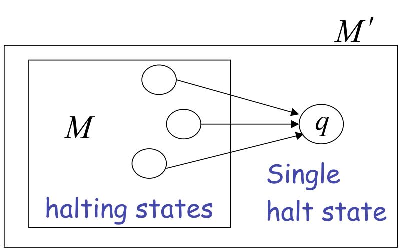


We need to convert one problem instance to the other problem instance



Convert M to M' :

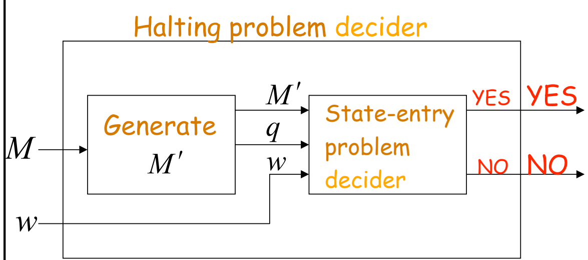
- Add new state q
- From any halting state of M add transitions to q



M halts on input w

if and only if

M' halts on state q on input w



We reduced the halting problem to the state-entry problem

Since the halting problem is undecidable, the state-entry problem is undecidable

END OF PROOF

Another example:

the halting problem

is reduced to

the blank-tape halting problem

The blank-tape halting problem

Input: Turing Machine M

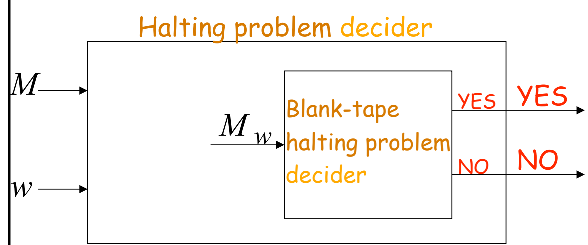
Question: Does M halt when started with a blank tape?

Theorem:

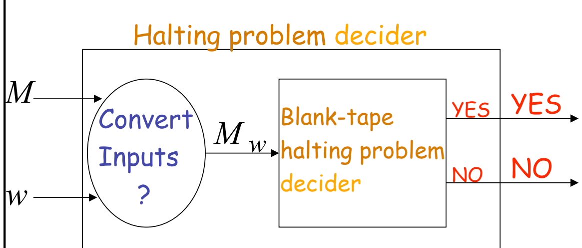
The blank-tape halting problem is undecidable

Proof: Reduce the halting problem to the blank-tape halting problem

We want to reduce the halting problem to the blank-tape halting problem:



We need to convert one problem instance to the other problem instance



What's Next

- Read
  - Linz Chapter 1,2.1, 2.2, 2.3, (skip 2.4), 3, 4, 5, 6.1, 6.2, (skip 6.3), 7.1, 7.2, 7.3, (skip 7.4), 8, 9, 10, 11, 12
  - JFLAP Chapter 1, 2.1, (skip 2.2), 3, 4, 5, 6, 7, (skip 8), 9, (skip 10), 11
- Next Lecture Topics
  - No Lecture on Thurs 11/20
- Quiz 4 in Recitation on Wednesday 12/3
  - Covers Linz 9, 10 and JFLAP 9, 11
  - Closed book, but you may bring one sheet of 8.5 x 11 inch paper with any notes you like.
  - Quiz will take the full hour
- Homework
  - Homework Due Tuesday After Thanksgiving