LIMITS OF ALGORITHMIC COMPUTATION

Chapter 12

1

Learning Objectives

- Explain and differentiate the concepts of *Computability* and *Decidability*.
- Define the Turing Machine *Halting problem*.
- Discuss the relationship between the *Halting problem* and *Recursively Enumerable Languages.*
- Give examples of *Undecidable problems* regarding *Turing Machines* to which the halting problem can be reduced.
- Give examples of *Undecidable problems* regarding *Recursively Enumerable Languages*.
- Determine if there is a solution to an instance of the **Post correspondence problem**.
- Give examples of *Undecidable problems* regarding *Context-Free Languages.*

2

Computability and Decidability

- Are there problems which are clearly and precisely stated, yet have *no algorithmic solution*?
- As stated in Chapter 9, a function *f* is *computable* if there exists a Turing Machine that computes the value of *f* for all arguments in its domain:

  i.e. $\exists$ a TM that computes $f(x)$, $\forall x \in \text{Dom}(f)$
- Since there may be a TM that can compute *f* for part of the domain, it is crucial to *define the domain* of *f* precisely: i.e. $\exists$ a TM that computes $f(x)$, $\forall x \in X \subset \text{Dom}(f)$.

  i.e. $q_0 x \vdash^*_M q_f f(x)$, $q_f \in F \quad \forall x \in X \subset Dom(f)$
- The concept of *decidability* applies to computations that result in a "*yes*" or "*no*" answer: a problem is *decidable* if there exists a TM that gives the correct answer for every instance in the domain.

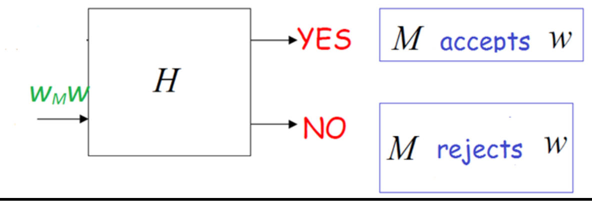
3

A simple Undecidable Problem: The Membership Problem

- The Membership Problem:
  - Input: A TM M and an input string w.
  - Question: Does M accept w? i.e. $w \in L(M)$? or $w \notin L(M)$?
- Theorem: The Membership Problem is Undecidable.

  There exists a TM M and w for which we can not decide/solve whether $w \in L(M)$, i.e. can't answer yes or no for $w \in L(M)$.

Proof by Contradiction) Assume that Membership problem is decidable. So, there exists a TM H that decides/solves it.



4

An Undecidable Problem: The Membership Problem

- Theorem: The Membership Problem is Undecidable.
  There exists a TM M and *w* for which we can not decide/solve whether $w \in L(M)$, i.e. can't answer yes or no for $w \in L(M)$.

Proof by Contradiction: cont.)

Let L be a recursively enumerable language.

Let M be the TM that accepts L: L(M) = *L*.

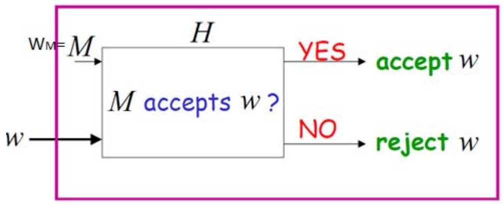
Let's try to prove L is also recursive (i.e. Turing-decidable):

We'll describe a TM H that accepts L and halts on any input.

Thus, L is recursive.

Note:

w_M = <M>: encoding of M.



5

An Undecidable Problem: The Membership Problem

- Theorem: The Membership Problem is Undecidable.
  There exists a TM M and w for which we can not decide/solve whether $w \in L(M)$, i.e. can't answer yes or no for $w \in L(M)$.

Proof by Contradiction: cont.)

Since L is chosen arbitrarily, we've proven that every recursively enumerable language is also recursive as H decides *w* in L(M).

BUT, there are recursively enumerable languages which are not recursive. Contradiction!!

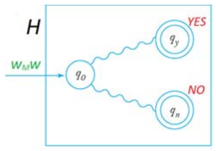
Thus, L is not recursive, i.e. There is not such a TM H.

Therefore, The membership problem is undecidable. Q.E.D.

6

The Turing Machine Halting Problem

- Definition 12.1: Let w_M be a string that describes a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$, and let a string $w \in \Sigma^*$.
  Assume that w_M and w are *encoded* as a string of 0's and 1's (ref. Sec.10.4)
  A (solution of the) *Halting problem* is a (Universal) TM H, which for any input w_M and w, performs the computation
  $q_0 w_M w \vdash^* x_1 q_y x_2$ if M applies to w *halts*, and
  $q_0 w_M w \vdash^* y_1 q_n y_2$ if M applies to w *does not halt*
  where $q_y, q_n \in F_H$ (final states of H)



- Many textbooks use the notations
  <M> for w_M : an encoding of TM M as $\{0, 1\}^+$
  <*w*> for w : an encoding of a string w as $\{0, 1\}^*$

7

The Turing Machine *Halting Problem*

- The Turing machine *Halting problem* can be stated as:

  Given the description/encoding of a Turing Machine M and an input string w, does M perform a computation that eventually halts on w?

- The *domain* of the problem is:
  the *set of all Turing machines*
  and *all input strings w*.
- Any attempts to simulate the computation on a *Universal Turing Machine* face the problem of *not knowing* if/when M has entered an *infinite loop*.

8

The TM Halting Problem (cont.)

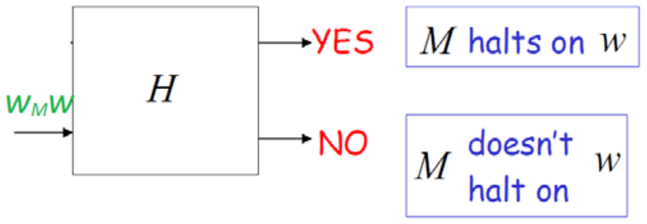
- Theorem 12.1: There does *not* exist any TM H that *decides* whether M halts on the input w;

  The Halting problem is therefore undecidable.

Proof by Contradiction)

Assume that there exists a TM, H, that solves the halting problem. The input to H is the string $w_M w$ (=<M><w>).

Then, given any $w_M w$, the TM H has to halt with an answer, either *yes* or *no*.



9

The TM Halting Problem (cont.)

- Theorem 12.1: There does *not* exist any TM H that *decides* whether M halts on the input *w* (required in Def. 12.1);

  The Halting problem is therefore undecidable.

Proof by Contradiction: cont.)

Construction of H:

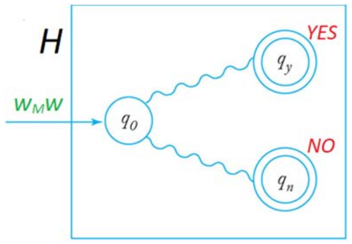
we achieve this if H halt in one of final states, q_y or q_n ,

when M halts on *w* or if M does not halt on *w*, respectively.

As in Definition, we want H to operate according to the following rules:

$q_0 w_M w \vdash_H^* x_1 q_y x_2$ if *M halts on w*

$q_0 w_M w \vdash_H^* y_1 q_n y_2$ if *M does not halt on w*



10

The TM Halting Problem (cont.)

- Theorem 12.1: There does *not* exist any TM H that *decides* an input instance (required in Def. 12.1);

  The Halting problem is therefore undecidable.

Proof: cont.)

Construction of TM H' from H:

Idea: If H answers *YES*, then loop forever.

If H answers *NO*, then halt.

Add the new states q_a and q_b to H'.

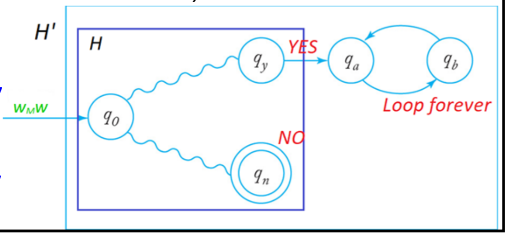
H' will enter an infinite loop where H reaches q_y and halts.

The action of H' is described by:

$q_0 w_M w \vdash_{H'}^* \infty$ if *M halts on w*

$q_0 w_M w \vdash_{H'}^* y_1 q_n y_2$

if *M does not halt on w*



11

The TM Halting Problem (cont.)

- Theorem 12.1: There does *not* exist any TM H that *decides* an input instance; *The Halting problem is therefore undecidable*.

Proof: cont.) Construction of TM $\widehat{H}$ from H':

Input: w_M (an encoding of TM *M*)

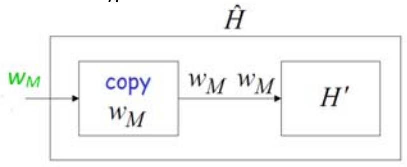
Idea: If *M* halts on input w_M then loop forever / else halt.

The new TM $\widehat{H}$ takes as input w_M and copies it, ending in its initial state q_0. After that, it behaves exactly like *H'*.

Then the action of $\widehat{H}$ is s.t.

$q_0 w_M \vdash_{\hat{H}}^* q_0 w_M w_M \vdash_{\hat{H}}^* \infty$ if *M halts on* w_M

$q_0 w_M \vdash_{\hat{H}}^* q_0 w_M w_M \vdash_{\hat{H}}^* y_1 q_n y_2$ if *M does not halt on* w_M.



12

The TM Halting Problem (cont.)

- Theorem 12.1: There does *not* exist any TM H that *decides* an input instance; *The Halting problem is therefore undecidable*.

Proof: cont.) Run TM $\widehat{H}$ with input itself:

Input: $\widehat{w_H}$ *(an encoding of TM* $\widehat{H}$)

Idea: If $\widehat{H}$ halts on $\widehat{w_H}$ then loop forever / else halt.

Since $\widehat{H}$ is a TM, $\widehat{H}$ has a description(i.e. encoding) in $\{0,1\}^+$, $\widehat{w_H}$.

Run $\widehat{H}$ with the input $\widehat{w_H}$. What happens?

If $\widehat{H}$ halts on $\widehat{w_H}$, then loop forever / else halts.

By identifying M with $\widehat{H}$, we get

$q_0\widehat{w_H} \vdash_{\hat{H}}^* \infty$ if $\widehat{H}$ *halts on* $\widehat{w_H}$

$q_0\widehat{w_H} \vdash_{\hat{H}}^* y_1 q_n y_2$ if $\widehat{H}$ *does not halt on* $\widehat{w_H}$. *Contradiction!*

Thus, there does ***NOT*** exist a TM H that decides 'yes' or 'no'

s.t. a TM halts on an input or not.

Therefore, the *Halting problem* is *UNDECIDABLE*!! Q.E.D.

13

The Halting Problem and Recursively Enumerable Languages

- Theorem 12.2: If the Halting problem *were decidable*, then every *Recursively Enumerable language* would be *Recursive*.

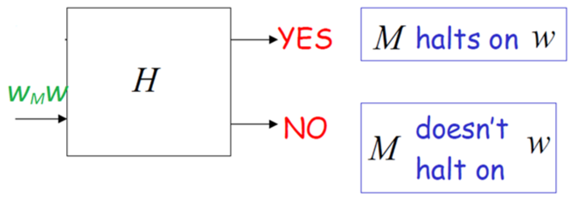
The 2nd way of Proof of the Halting problem:

If L = $\{w_M w \mid M$ halts on $w\}$ were recursive,

then every RE language is recursive.

Proof by Contradiction) Assume the halting problem is decidable.

Then, there exists a TM H that solves the halting problem.



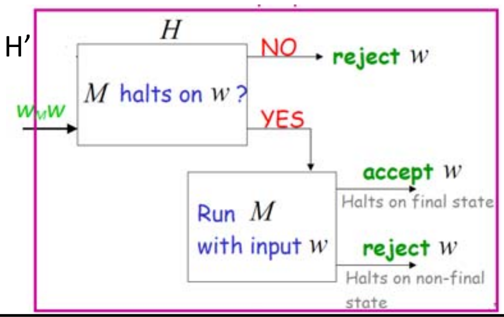
14

The Halting Problem and Recursively Enumerable Languages

- Theorem 12.2: If the Halting problem *were decidable*, then every *RE language* would be *Recursive*.

Proof by Contradiction: cont.) Let L be a recursively enumerable language and M be the TM that accepts L: L = L(M)

We will describe a TM H' that accepts L and halts on any input w, proving that L is also recursive.



15

The Halting Problem and Recursively Enumerable Languages

- Theorem 12.2: If the Halting problem *were decidable*, then every *RE language* would be *Recursive*.

Proof) Let H be a TM that decides the halting problem.

Then, we can apply H to the accepting TM M, $w_M w$.

- If H says '*no*', i.e. M does not halt on w, then by definition of L, the input string *w* is not in L, $w \notin L$.
- If H says '*yes*', i.e. M *halts*, then M will decide if $w \in$ L or $w \notin$ L.

Thus, *L* is recursive by the above *membership algorithm* for *L*.

BUT, we know that there exists some R.E. languages that are *not Recursive*. Contradiction!!

Thus, the contradiction implies such a TM H cannot exist, i.e. the halting problem is undecidable. Q.E.D.

16

Reducing One Undecidable Problem to Another

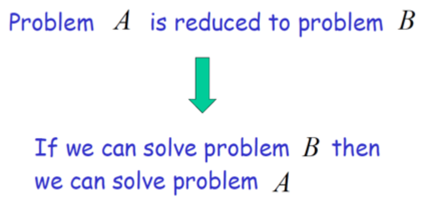
- Two main techniques for showing the problems are undecidable: *Diagonalization* and *Reduction*.
- The halting problem was connected to the membership problem: a technique of *reduction*.
- A problem A is *reduced* to a problem B

  if the decidability of A follows

  from the decidability of B.

i.e. if B is decidable,

A is decidable.

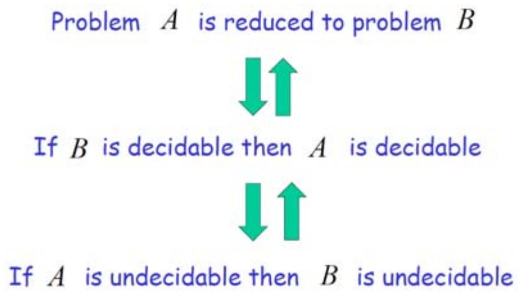

Problem A is reduced to problem B

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

17

Reducing One Undecidable Problem to Another

- A is reducible to B if we can use B as a *sub-algorithm* to solve A.
- Use a halting TM for A in the construction of a halting TM that decides problem B.


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.
- So, if the halting problem is undecidable,

  then the state-entry problem is undecidable.

18

Reduction (from Goddard's book)

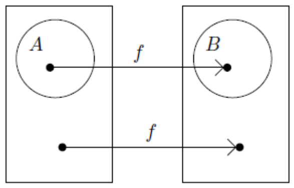
By our Definition 9.4, a function $f : \Sigma^* \rightarrow \Sigma^*$ is *Turing-computable* if there exists a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ s.t.

$$q_0 w \vdash^*_M q_f f(w),\ q_f \in F \quad \forall w \in D.$$

Definition: Reduction

Let A, B be languages.

We say that A is *reducible* to B, written $A \leq_m B$, if there exists a Turing-computable function f s.t. $w \in A$ exactly when $f(w) \in B$.



19

Reduction preserve Hardness (from Goddard's book)

Fact:

a) If A is reducible to B and B is recursive/decidable, then A is recursive/decidable.

b) If A is reducible to B and A is not recursive/undecidable, then B is not recursive/undecidable.

Notation $\leq$:

The above fact shows if one writes $A \leq_m B$,

then *B is as least as hard as A*. For example:

Fact: For any languages A, B, and C,

if $A \leq_m B$, $B \leq_m C$, then $A \leq_m C$.

If a function f reduces A to B and a function g reduces B to C, then a function h defined by $h(w) = g(f(w))$ reduces A to C.

20

State Entry Problem (SE): Undecidable

- The *State-Entry Problem:*

  Input: *TM M, state q, string w.*

  Question: *Does M enter a state q on input w?*

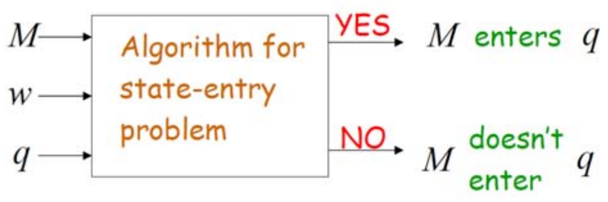
- Given any TM M and string *w*, *decide* whether or not the state *q* is ever entered when *M* is applied to *w*.
- If we had an algorithm that solves the state-entry problem, it could be used to solve the halting problem.
- However, because the halting problem is undecidable, the state-entry problem must also be undecidable.

21

State Entry Problem: Undecidable

- Theorem: The State-Entry (SE) problem is undecidable.

Proof)
Reduce the halting problem to the state-entry problem.
Suppose we have an algorithm (TM) that solves the SE prob.
Then, we can construct an algorithm (TM) that solves the halting problem from the algorithm of SE problem.

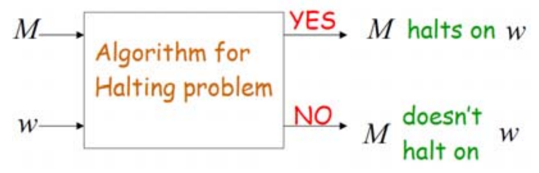


22

State Entry Problem: Undecidable

- Theorem: The State-Entry (SE) problem is undecidable.

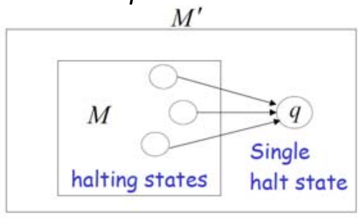
Proof) We want to design the algorithm (TM M) for the halting problem.



Modify any machine M to construct M' :

- Add a new state *q*.
- From any halting state, add transitions to *q*.

M halts iff M' halts on state *q*.



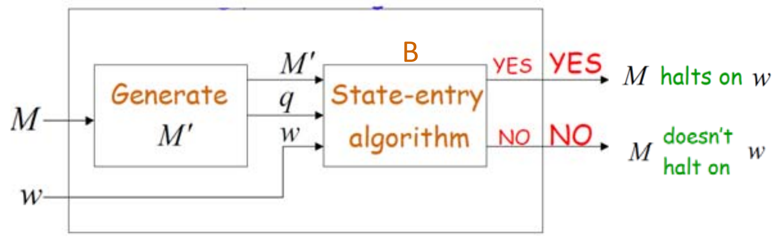
23

State Entry Problem: Undecidable

- Algorithms (TM) for Halting Problem:

Inputs: TM *M* and string *w*.

1. Construct a TM *M'* with state *q*.
2. Run algorithm for SE-problem with inputs: *M', q, w*.



- We reduced the halting problem to the SE problem.
- Since the halting problem is undecidable, it must be that the state-entry problem is also undecidable. Q.E.D.

24

Reducing One Undecidable Problem to Another

- Example 12.1: the *State-Entry Problem (SE problem)*

Given any TM M = (Q, Σ, Γ, δ, q_0, □, *F*) and any $q \in Q$, $w \in \Sigma^+$,

decide whether or not the state *q* is ever entered when *M* is applied to *w*.

This problem is undecidable.

To reduce the Halting problem to the SE-Problem, suppose we have an algorithm B that solves the SE-problem. Then, we can use it to solve the halting problem.

For example, given any *M* and *w*, we first modify M to get M' as follows:

M' halts in state *q* if and only if M halts.

It can be done by looking at the transition function δ of M.

If M halts, it does so because some $\delta(q_i, a)$ is undefined.

To get M', we change every such *undefined* δ to $\delta(q_i, a) = (q, a, R)$,

where *q* is a final state. We apply the SE-algorithm A to (M', *q, w*).

If A answers '*yes*', i.e., the state *q* is entered, then (*M, w*) halts.

If A answers '*no*', then (*M, w*) does not halt.

Thus, the assumption the SE-problem is decidable gives us an algorithm for the halting problem. Because the halting problem is undecidable, the SE-problem must also be undecidable.

25

The Blank-Tape Halting Problem(BTH): Undecidable

- The halting problem is reduced to the blank-tape halting problem.
- The blank-tape halting problem (BTH):
  - Input: TM M
  - Question: Does M halt when started with a blank tape?
- Theorem: The Blank-Tape Halting problem is undecidable.

Proof by Reduction) Reduce the halting problem
to the blank-tape halting problem.

Suppose we have an algorithm for the BTH problem.

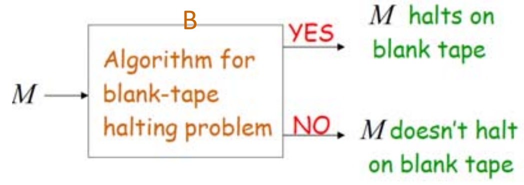
We'll construct an algorithm for the halting problem.

26

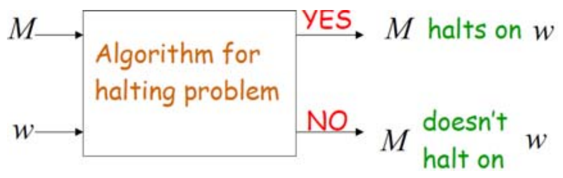
The Blank-Tape Halting Problem: Undecidable

Proof: cont.)

- The algorithm for the BTH problem:


B
Algorithm for blank-tape halting problem
M
YES
M halts on blank tape
NO
M doesn't halt on blank tape

- Design a TM that decides the halting problem.

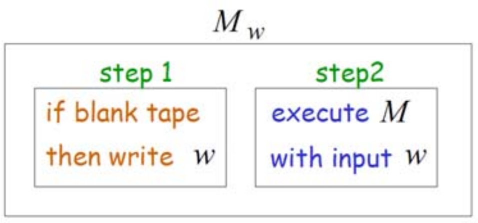


27

The Blank-Tape Halting Problem: Undecidable

Proof: cont.)

- Construct a new TM M_w:
  - On blank tape, writes w.
  - Then, continues execution like M.



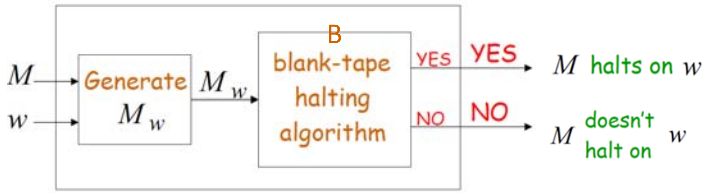
- M halts on input string *w* if and only if M_w halts when started with blank tape.

28

The Blank-Tape Halting Problem: Undecidable

Proof: cont.)

- Algorithm for halting problem:
  - Inputs: TM M and string w
  1. Construct M_w .
  2. Run algorithm for BTH-problem with input M_w



- We reduced the halting problem to the BTH-problem.
- Since the halting problem is undecicable, the blank-tape halting problem is also undecidable. Q.E.D.

29

Undecidable Problems for Recursively Enumerable Languages

- There is *no membership algorithm* for Recursive Enumerable languages.
- RE languages are so general that most related questions are undecidable.
- Usually, there is a way to reduce the halting problem to questions regarding recursively enumerable languages, such as
  - Is the language generated by an unrestricted grammar empty?
  - Is the language accepted by a Turing machine finite?

30

Undecidable Problems for Recursively Enumerable Languages

Rice Theorem:

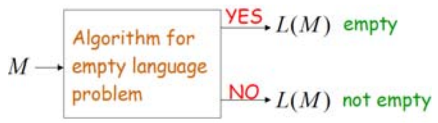
Every nontrivial property of a RE language is *undecidable*.

The "*nontrivial*" refers to a property possessed by *some* RE languages, not by all RE languages.

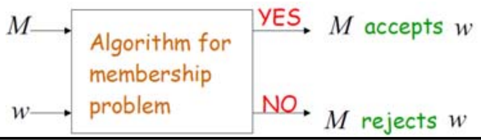
31

"L(G) = ∅" Problem: Undecidable

- Theorem 12.3: Let G be an unrestricted grammar.
  For any RE languages L, s.t. L=L(G), the problem to determine whether "L(G) = ∅" is undecidable.
- Proof) Reduce the membership problem for RE-lang. to this prob.
  Let M be the TM that accepts L : L(M) = L.
  Assume we have the empty language algorithm:



Let's design the membership algorithm:



32

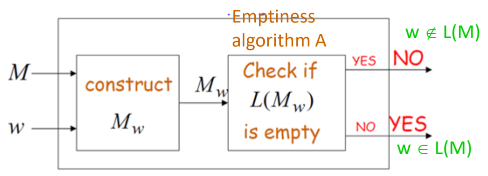
“L(G) = ∅” Problem: Undecidable (cont.)

- Theorem 12.3: Let G be an unrestricted grammar.
  For any RE languages L, s.t. L=L(G), the problem to determine whether “L(G) = ∅” is undecidable.
- Proof: cont.) First, construct TM M_w:
  M saves its input on a special part of its tape.
  When M enters a final state, compare original input string with w.
  Accept *w* if the original input = w.
  $w \in L$ if and only if $L(M_w) \neq \varnothing$ because $L(M_w) = L(M) \cap \{w\} = \{w\}$.
  We can do this by changing δ of M, creating M_w for each *w*
  s.t. $L(M_w) = L(M) \cap \{w\}$

33

“L(G) = ∅” Problem: Undecidable (cont.)

- Algorithm for membership problem:
  Inputs: TM M and string w.
  1. Construct M_w.
  2. Determine if $L(M_w) = \varnothing$.
     YES: then $w \notin L(M)$
     NO: then $w \in L(M)$.



- It gives us a membership algorithm for any recursively enumerable language.
- But it contradicts previous results that there is no such membership algorithm, i.e. undecidable.
- So, the Emptiness problem for RE language with an unrestricted grammar is undecidable.

34

Is the Language Accepted by a TM finite?

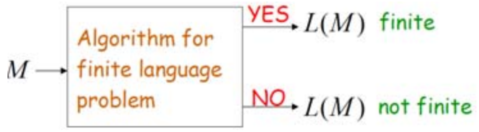
- Theorem 12.4: Given a TM M, the problem of determine whether or not *L*(*M*) is finite is undecidable. ⇔ For a RE language L,

  the problem to determine if L is finite is undecidable.

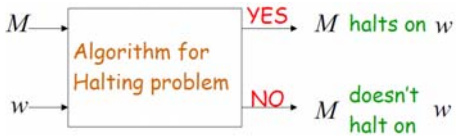
Proof) Let's Reduce the halting problem to this problem.

Let M be the TM that accepts L : L(M) = L.

Assume we have the finite language algorithm:



We'll design the halting problem algorithm:



35

Is the Language Accepted by a TM finite?

- Theorem 12.4: Given a TM M, the problem of determine whether or not *L*(*M*) is finite is undecidable. ⇔ For a RE language L,

  the problem to determine if L is finite is undecidable.

Proof: cont.). First, Construct a TM M_w:

Initially, simulates M on the input w.

If M enters a halt state (i.e. M halts on w),

accept any input (infinite language).

Otherwise, accept nothing (finite language).

M halts on *w* if and only if $L(M_w)$ is not finite.

36

Is the Language Accepted by a TM finite?

- Theorem 12.4: Given a TM M, the problem of determine whether or not $L(M)$ is finite is undecidable. $\Leftrightarrow$ For a RE language L, the problem to determine if L is finite is undecidable.

Proof: cont.).

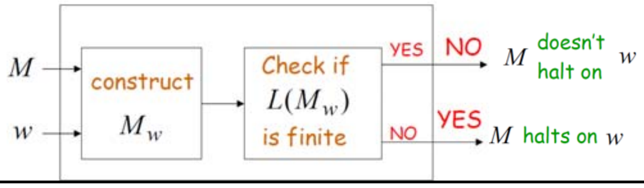
Algorithm for Halting problem:

Inputs: TM M and string w.

1. Construct M_w.
2. Determine if $L(M_w)$ is finite

YES: then M doesn't halt on w.

NO: then M halts on w.



37

The Undecidability of the "L(M) is Finite" Problem

Cont.)

- It gives us an algorithm for the halting problem.
- But it contradicts previous results that there is no such algorithm for the halting problem, i.e. undecidable.
- So, the Finiteness problem for RE is undecidable.

38

The Post Correspondence Problem

- Some *undecidable* problems for Context-Free Languages:
  - Is CFG G ambiguous?
  - Is $L(G_1) \cap L(G_2) = \varnothing$?
- We need a tool to prove those problems for CFL are *undecidable*.
- The Post Correspondence Problem (PC-Problem):
  - Input: Two sequences of n strings on an alphabet Σ, e.g.

    $A = w_1, w_2, \ldots, w_n$ and $B = v_1, v_2, \ldots, v_n$
  - There is a Post Correspondence Solution for the pair (*A, B*) if there is a *nonempty sequence* of integers $i, j, \ldots, k,$ s.t. $w_i w_j \ldots w_k = v_i v_j \ldots v_k$.

39

The Post Correspondence Problem

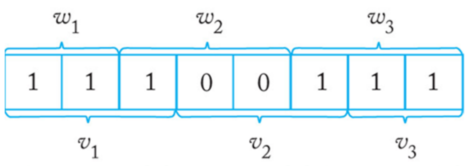
- Example 12.5: Let $\Sigma = \{0, 1\}$ and A and B consist of

  $A = w_i$'s where $w_1 = 11, w_2, = 100, w_3 = 111$ and

  $B = v_i$'s where $v_1 = 111, v_2, = 001, v_3 = 11$

A PC solution for this instance of (A, B): 1, 2, 3.

$w_1 w_2 w_3 = v_1 v_2 v_3 = 11100111$



- Example: A: $w_1 = 00, w_2, = 001, w_3 = 1000$ and

  B: $v_1 = 0, \quad v_2, = 11, \quad v_3 = 011.$

There is no solution because |B| < |A|.

40

The Undecidability of the Post Correspondence Problem

- The Modified Post Correspondence Problem (MPC).
  - Input: $A = w_1, w_2, \ldots, w_n$ and $B = v_1, v_2, \ldots, v_n$
  - MPC-Solution: a *nonempty sequence* of integers $i, j, \ldots, k,$ s.t. $w_1 w_i w_j \ldots w_k = v_1 v_i v_j \ldots v_k$.
- Example:
  - A: $w_1 = 11$, $w_{2,} = 111$, $w_3 = 100$ and

    B: $v_1 = 111$, $v_{2,} = 11$, $v_3 = 001$.
  - MPC-solution: 1, 3, 2.

    $w_1 w_3 w_2 = v_1 v_3 v_2 = 11100111$

1. We'll prove that MPC-problem is Undecidable.
2. We'll prove that PC-problem is undecidable.

41

The Undecidability of the Post Correspondence Problem

Claim 1 (Theorem 12.6): The MPC-problem is Undecidable.

Proof) Reduce the membership problem to the MPC-problem.

Membership Problem for Recursive language: Undecidable

$\begin{cases} \text{Input:} & \text{Recursive language } L \text{ and a string } w. \\ \text{Question:} & w \in L \text{ or } w \notin L? \end{cases}$

Membership Problem for Unrestricted grammar/RE language: Undecidable

$\begin{cases} \text{Input:} & \text{Unrestricted Grammar } G \text{ and a string } w. \\ & \text{(i. e. RE language with TM M, L=L(M)=L(G)} \\ \text{Question:} & w \in L(G) \text{ or not?} \end{cases}$

The reduction of the membership problem to the MPC problem:

For Unrestricted grammar G and string w,

we construct a pair A, B, such that

A, B has an MPC-solution if and only if $w \in L(G)$.

With G and w, create the pair (A, B) in the table.

42

The Undecidability of the PC-Problem

w_i A	v_i B	
w_1 $FS \Rightarrow$	v_1 F	F is a symbol not in $V \cup T$
a	a	for every $a \in T$
V_i	V_i	for every $V_i \in V$
E	$\Rightarrow wE$	E is a symbol not in $V \cup T$
y_i	x_i	for every $x_i \to y_i$ in P
$\Rightarrow$	$\Rightarrow$	

The order of the rest of the strings is immaterial.
We want to claim eventually that w $\in$L(G)
iff the sets A and B constructed in this way have an MPC solution.

43

The Undecidability of the PC-Problem

- Example 12.6:

Grammar G: $S \to aABb \mid Bbb$

$Bb \to C,\ AC \to aac$

String $w = aaac$

i	w_i in A	v_i in B
1	$FS \Rightarrow$	F
2	a	a
3	b	b
4	c	c
5	A	A
6	B	B
7	C	C
8	S	S
9	E	$\Rightarrow aaacE$
10	$aABb$	S
11	Bbb	S
12	C	Bb
13	aac	AC
14	$\Rightarrow$	$\Rightarrow$

$aaac \in L(G)$

$S \Rightarrow aABb \Rightarrow aAC \Rightarrow aaac$

44

The Undecidability of the Post Correspondence Problem

- Example 12.6 (cont.):

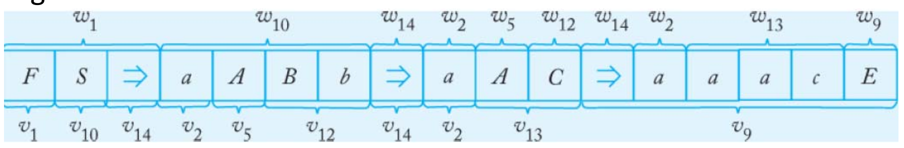
Grammar G: S $\rightarrow$ *aABb* | *Bbb,*

Bb $\rightarrow$ *C*, *AC* $\rightarrow$ *aac*

String *w = aaac*

S $\Rightarrow$ *aABb* $\Rightarrow$ *aAC* $\Rightarrow$ *aaac* $\in$ *L(G)*

How does this derivation is paralleled by an MPC solution with fig12.9?



MPC-Solution of (A, B): 1, 10, 14, 2, 5, 12, 14, 2, 13, 9

Theorem 12.5:

(A, B) has an MPC-solution iff *w* $\in$ *L(G).*

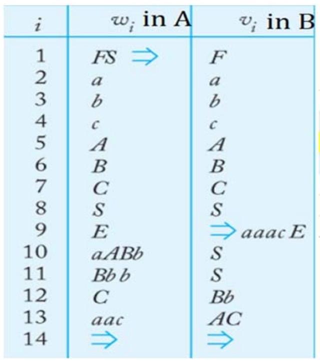
45

The Undecidability of the Post Correspondence Problem

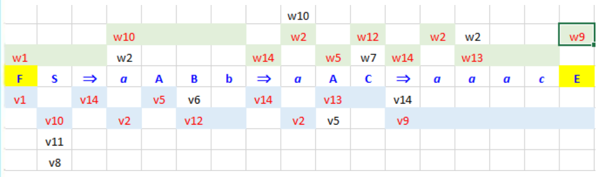
- Example 12.6 (cont.):

G: S $\rightarrow$ *aABb* | *Bbb, Bb* $\rightarrow$ *C*, *AC* $\rightarrow$ *aac*

w = aaac *S* $\Rightarrow$ *aABb* $\Rightarrow$ *aAC* $\Rightarrow$ *aaac* $\in$ *L(G)*



i	w_i in A	v_i in B
1	$FS \Rightarrow$	F
2	a	a
3	b	b
4	c	c
5	A	A
6	B	B
7	C	C
8	S	S
9	E	$\Rightarrow aaacE$
10	$aABb$	S
11	Bbb	S
12	C	Bb
13	aac	AC
14	$\Rightarrow$	$\Rightarrow$



								w10								
			w10					w2		w12		w2	w2			w9
w1			w2				w14		w5	w7	w14		w13			
F	S	⇒	a	A	B	b	⇒	a	A	C	⇒	a	a	a	c	E
v1		v14		v5	v6		v14		v13		v14					
	v10		v2		v12			v2	v5		v9					
	v11															
	v8															

46

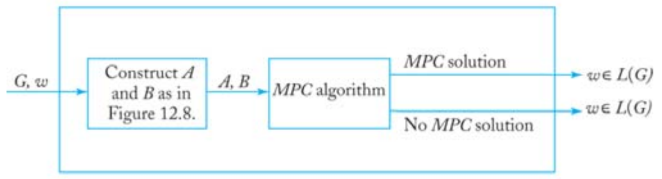
The Undecidability of the Post Correspondence Problem

- Algorithm for Membership Problem:

  Input: Unrestricted grammar G and string w.

  Construct the pair A, B.

  If A, B has an MPC-solution then *$w \in L(G)$ / else $w \notin L(G)$.*



An algorithm for construction A and B from G and w exists, but a membership algorithm for G and w does not (undecidable). Thus, there doesn't exist any algorithm to decide MPC-problem. Therefore, *MPC-problem is undecidable.* *Q.E.D.*

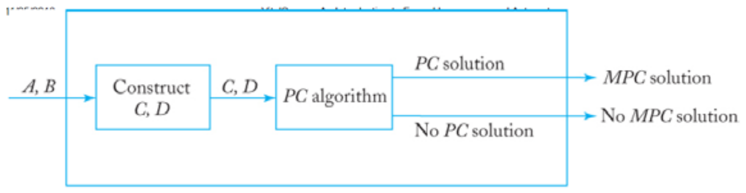
47

The Undecidability of the Post Correspondence Problem

- Claim 2 (Theorem 12.7): The PC problem is Undecidable.

Proof) Reduce the MPC-problem to the PC-problem.

See the textbook.



48

The Undecidability of the Post Correspondence Problem

- Theorem 12.7: There is *no algorithm* to decide if a solution sequence exists under all circumstances, so the *Post Correspondence problem is undecidable.*
- Although a proof of theorem 12.7 is quite lengthy, this very important result is crucial for showing the *undecidability of various problems involving Context-Free Languages.*

49

Undecidable Problems for CFL

- The Post Correspondence Problem is a convenient tool to study some questions involving Context-Free languages.
- The following questions, among others, can be shown to be *undecidable.*
  - Given an arbitrary Context-Free Grammar G, is G *ambiguous*?
  - Given arbitrary CFG G_1 and G_2, is $L(G_1) \cap L(G_2) = \varnothing$?
  - Given arbitrary CFG G_1 and G_2, is $L(G_1) = L(G_2)$?
  - Given arbitrary CFG G_1 and G_2, is $L(G_1) \subseteq L(G_2)$?

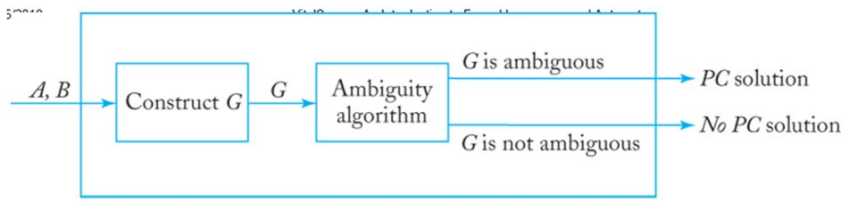
50

Undecidable Problems for CFL

- Theorem 12.8: There exists no algorithm for deciding whether any given CFGs is *ambiguous*.

Proof) See the textbook.

Reduce PC-problem to the Ambiguity problem of CFG.



51

Undecidable Problems for CFL

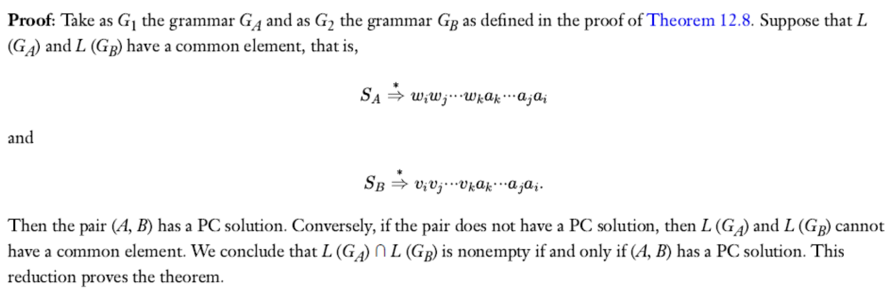
- Theorem 12.9:

There exists no algorithm for deciding whether or not

$L(G_1) \cap L(G_2) = \varnothing$ for given arbitrary CFG G_1 and G_2.

Proof) See the textbook.

Reduce PC-problem to this Emptiness of two CFLs of their CFGs.



Proof: Take as G_1 the grammar G_A and as G_2 the grammar G_B as defined in the proof of Theorem 12.8. Suppose that $L(G_A)$ and $L(G_B)$ have a common element, that is,

$$S_A \overset{*}{\Rightarrow} w_i w_j \cdots w_k a_k \cdots a_j a_i$$

and

$$S_B \overset{*}{\Rightarrow} v_i v_j \cdots v_k a_k \cdots a_j a_i.$$

Then the pair (A, B) has a PC solution. Conversely, if the pair does not have a PC solution, then $L(G_A)$ and $L(G_B)$ cannot have a common element. We conclude that $L(G_A) \cap L(G_B)$ is nonempty if and only if (A, B) has a PC solution. This reduction proves the theorem.

52