Chapter 5 Undecidability

Elements Of The Theory Of Computation
Zhejiang University/CS/Course/Xiaogang Jin
E-Mail: xiaogangj@cise.zju.edu.cn

5.1 Church-Turing Thesis

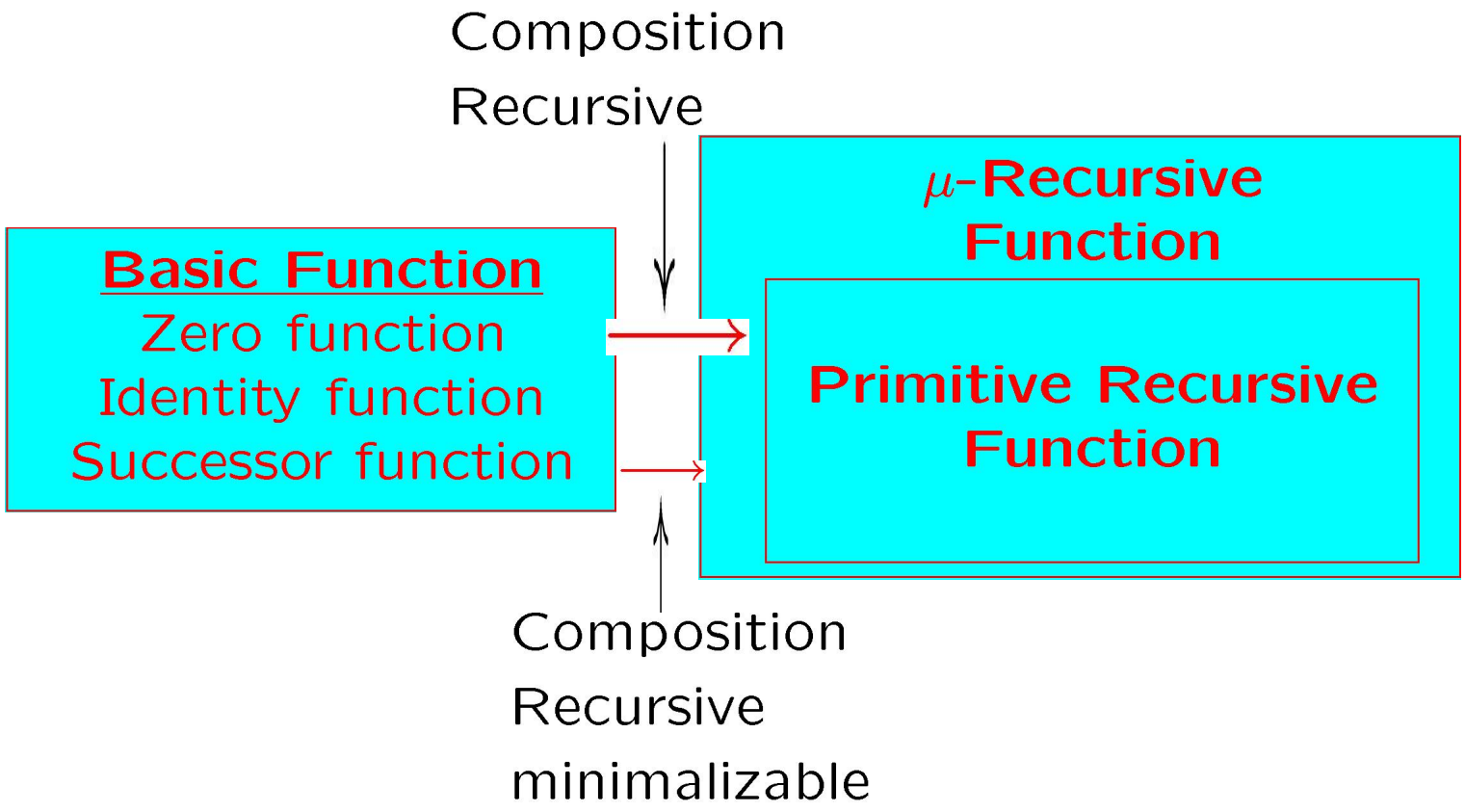
Question: What can be computed?

- **Turing Macine**

  – Lots of models: λ-calculus, mechanical devices, Post-Turing System , Grammar, $\cdots$

  – All of these seem to define the same set of functions

- **Gödel' Recursive Functions**



- **Church - Turing thesis:**

Intuitive notation of "computable"
=
Formal notation of "computable functions by TM"

Remark:

- Church-Turing thesis, not a theorem can not be proved but could be disproved at some future date.

- An algorithm is a program that always halts (with the correct answer), so it is recursive.

- A problem is decidable if it is recursive. If not, it is undecidable.

5.2 Universal Turing Machines

□ **Turing Machines**

– is an "unprogrammable" piece of hardware, specialized at solving one particular problem.

– is a software, seem to be able to carry out very general computations.

□ **The ultimate test:** can they be programmed?

– Can we design a "TM simulator" that is a TM?

– Can we design U that takes as input M and w, and produces the same result as runnning M on w?

Problem:

There is no bound on the number of states or symbols of M.

So U, which is a fixed machine, cannot have single symbols corresponding to the states or symbols of M

— it must have a fixed alphabet.

Solution:

Adopt an encoding of the states and symbols of M, and then a string representation of M itself.

Representation of Turing Machines

- **For $\sum \cup \{\leftarrow, \rightarrow\}$.**
  - Let $j = \min\{j \mid j \in \mathbb{Z} \text{ and } 2^j \geq |\sum| + 2\}$.

Each symbol in $\sum$ will be represented as the letter a followed by a string of j bits.

- We fix the representations of the special symbols $\sqcup$, $\triangleright$, $\leftarrow$, and $\rightarrow$ to be the lexicographically four smallest symbols, respectively:

$\sqcup$	$a0^j$
$\triangleright$	$a0^{j-1}1$
$\leftarrow$	$a0^{j-2}10$
$\rightarrow$	$a0^{j-2}11$

- **For states:**
  - Let $i = \min\{i \mid i \in \mathbb{Z} \text{ and } 2^i \geq |K|\}$.

Each state in K will be represented as the letter q followed by a binary string of length i.

  - Use $q0^i$ to represent the start state.

- **For the TM M itself:**
  - *A list, in increasing lexicographic order , starting with $\delta(s, \sqcup)$, of the encoded quadruples.*
  - *The set of halting states will be determined indirectly, by absence of its states as first components in any quadruple of "M". When M decides a language, and thus $H = \{y, n\}$, we convent that y is the lexicographically smallest of the two halt states.*

Example: Consider the Turing machine $M = (K, \sum, \delta, s, \{h\})$, where $K = \{s, q, h\}$, $\sum = \{\sqcup, \triangleright, a\}$ and δ is given in this table.

state	symbol	δ
s	a	$(q, \sqcup)$
s	$\sqcup$	$(h, \sqcup)$
s	$\triangleright$	$(s, \rightarrow)$
q	a	(s, a)
q	$\sqcup$	$(s, \rightarrow)$
q	$\triangleright$	$(q, \rightarrow)$

- Since $|K| = 3$ and $|\sum| = 3$, we have $i = 2$ and $j = 3$.

state/symbol	representation
s	$q00$
q	$q01$
h	$q11$
$\sqcup$	$a000$
$\triangleright$	$a001$
$\leftarrow$	$a010$
$\rightarrow$	$a011$
a	$a100$

The representation "M" of the TM is the following string:

$$\begin{aligned}\text{“}M\text{”} = &(q00, a100, q01, a000),\ (q00, a000, q11, a000),\\ &(q00, a001, q00, a011),\ (q01, a100, q00, a011),\\ &(q01, a000, q00, a011),\ (q01, a001, q01, a011).\end{aligned}$$

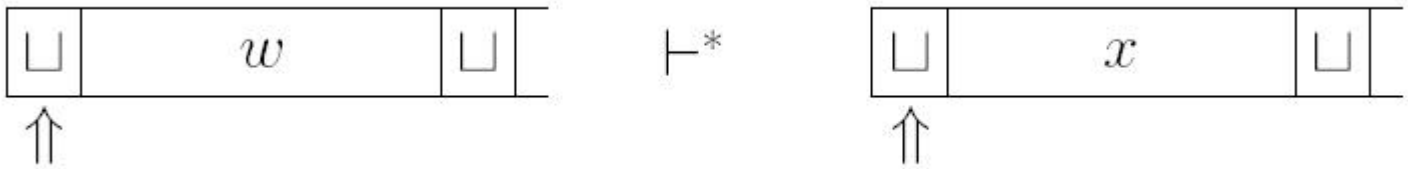
Universal Turing Machine

- The universal machine should give as the value of its computation on inputs M and w whatever value M would produce on input w (all duly encoded).

- U have the following property:
  U halts on input "M" "w" iff M halts on the input w. i.e.

$$U(\text{“}M\text{”}\ \text{“}w\text{”}) = \text{“}M(w)\text{”}.$$

If M



Then U

- Design of U
  - Construct a 3-tapes Turing Machine U':
    (U will be the standard TM that simulates U')

#1	Encoding of Tape of M
#2	Encoding of M
#3	Encoding of Current State of M

(1) On input "M" "w", copy "M" to the second tape , align "w" at the left end of the first tape, and write "s" on the third.

(2) Simulate one step of the computation by M by finding on the second tape the quadruple corresponding to the current state and scanned symbol of M, and changing the first and third tapes accordingly.

(3) Repeat step (2) until the third tape contains "h" (which may never happen).

5.3 The Halting Problem

- **The motivation:**

– to find a nonrecursive language.

- We know such languages exist, by a counting argument:
  - ☐ Every recursive language is decided by a TM
  - ☐ There are only countably many TMs
  - ☐ There are uncountably many languages

⇒Most languages are not recursive.

- **Question:**

Suppose you have written the following programs:

- $halts(P, X)$
  - Program P halts on input X, return "yes" .
  - Program P runs forever on input X, return "no" .

$diagonal(X)$:
a: if $halt(X, X)$ then goto a else $halt$

- Does $diagonal(diagonal)$ *halt?*

$diagonal(X)$:
a: if $halt(X, X)$ then goto a else $halt$

$diagonal(diagonal)$ halt **iff** halt $(diagonal, diagonal)$ =no.

Contradiction!

Conclusion:

There is no program, no algorithm for solving the halting problem:

—to tell whether arbitary programs would halt or loop.

Theorem: Let

$$H = \{\text{“}M\text{”}\ \text{“}w\text{”} : \text{TM } M \text{ halts on input string } w\}.$$

The language H is not recursive; therefore, the class of recursive languages ia a strict subset of the class of recursively enumerable languages.

Remark:

- H is the formalized version of the halting problem
- H is recursively enumerable
  - *because the universal TM semidecides* H $(L(U) = H)$

Proof:

I. H recursive $\Rightarrow$ H_1 recursive,

where $H_1 = \{\text{“}M\text{”} : M \text{ halts on input “}M\text{”}\}$.

If there were a TM M_0 that decide H, then the TM M_1 to decide H_1 as follows:

i) transforms its input tape from $\triangleright \sqcup$ “M”$\underline{\sqcup}$ to $\triangleright \sqcup$ “M” “M”$\underline{\sqcup}$
ii) simulates M_0 on this input.

II. H_1 recursive $\Rightarrow$ $\overline{H_1}$ recursive,

The class of recursive languages is closed under complement(P 199 Theorem 4.2.2).

$\overline{H_1} = \{\text{“}M\text{”} : M \text{ does not halt on input “}M\text{”}\}$

III. $\overline{H_1}$ is not even r.e. language.

- Suppose M^* semidecides $\overline{H_1}$
- Is “M^*” $\in \overline{H_1}$?

“M^*” $\in \overline{H_1} \Leftrightarrow$ “M^*” $\notin H_1$

$\Leftrightarrow M^*$ does not halt on “M^*”

M^* semidecides $\overline{H_1}$ so

“M^*” $\in \overline{H_1} \Leftrightarrow M^*$ halts on input “M^*”

$\Rightarrow$ so there is no such M^* **Contradiction**

Remark: This is a diagonalization argument.

- Look at H_1 as a table:

	"M_0"	"M_1"	"M_2"	"M_3"
M_0	Y	N	N	Y
M_1	Y	Y	N	N
M_2	N	N	N	N
M_3	Y	Y	Y	Y

- Entry matching (M_i, "M_j") is Y iff M_i halts on "M_j" .
- $\overline{H}_1$ is represented by the complement of the diagonal, which must be different from every row.

Plus strings that do not represent Turing Machines at all

Every r.e. language is recursive iff the particular r.e language H is recursive.

Suppose that H is indeed decided by some TM M_0. Given any particular TM M semideciding a language $L(M)$, then we could design a TM M' that decides $L(M)$ as follows:

i) transforms its input tape from $\triangleright \sqcup w \underline{\sqcup}$ to $\triangleright \sqcup$ "M" "w" $\underline{\sqcup}$
ii) simulates M_0 on this input.

Remark: There are **reductions** from all r.e. languages to H and H is **complete** for the r.e. languages.

Theorem: The class of recursively enumerable languages is not closed under complement.

Counterexample:

H_1, like H is recursively enumerable languages, but $\overline{H_1}$ is not recursively enumerable.

5.4 Undecidable Problems about TM

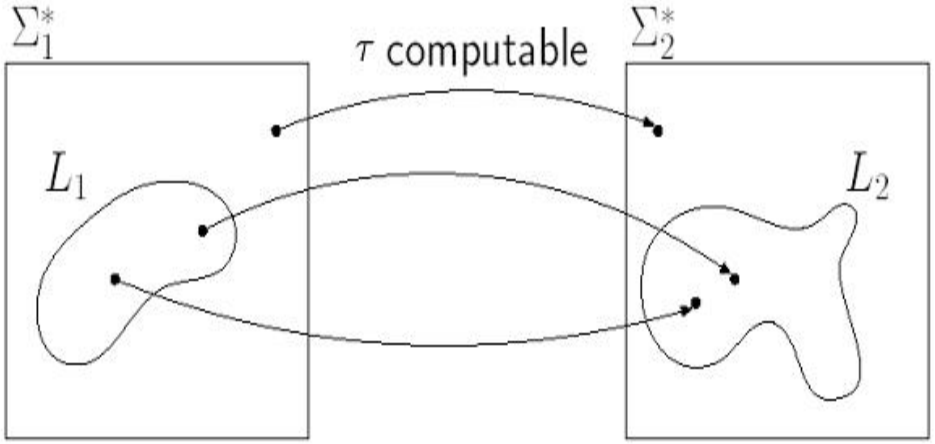
• Any algorithm can be turned into a TM that halts on all inputs.

• Problems for which no algorithms exist are called **undecidable** or **unsolvable**

• H is not recursive.

– The most famous undecidable problem is called **the halting problem for TM**.

Definition: Let $L_1, L_2 \subseteq \sum^*$ be languages. A **reduction** from L_1 to L_2 is a recursive function $\tau : \sum^* \rightarrow \sum^*$ such that $x \in L_1$ iff $\tau(x) \in L_2$.



Theorem: If L_1 is not recursive, and there is a reduction from L_1 to L_2, then L_2 is also not recursive.

Proof: Suppose that L_2 is recursive, say decided by TM M_2. Let T be the TM that computes the reduction τ. Then the TM TM_2 would decide L_1. But L_1 is undecidable.
— a contradiction.

Remark:

If there is a reduction of L_1 to L_2, then
– if L_2 is decidable, then so is L_1;
– if L_1 is undecidable, then so is L_2.

Theorem: The following problems about TM are undecidable.

(a) Given a TM M and an input w, does M halt on input w?

(b) Given a TM M, does M halt on the empty tape?

(c) Given a TM M, is there any string at all on which M halts?

(d) Given a TM M, does M halt on every input string?

(e) Given two TM M_1 and M_2, do they halt on the same input strings?

(f) Given a TM M is the language that M semidecides regular? Is it context-free? Is it recursive?

(g) Furthermore, there is a certain fixed machine M, for which the following problem is undecidable: Given w, does M halt on w?

Proof: (a) was proved in the prevous section.

(b) We describe a reduction from H to the language

$$L = \{\text{“}M\text{”} : M \text{ halts on } e\}$$

Given “M” and “w”.

Construct M_w, where M_w is a TM that writes w on the empty tape and then runs M.

If $w = a_1 a_2 \cdots a_n$, then M_w is:

$$Ra_1 Ra_2 R \cdots Ra_n L \sqcup M.$$

The function τ : “M” “w” $\longmapsto M_w$ is recursive.

(c) We can reduce the language $L = \{\text{“}M\text{”} : M \text{ halts on } e\}$ to $L' = \{\text{“}M\text{”} : M \text{ halts on some input}\}$ as follows.

Given “M”.

Construct M': Erases any input and then simulates M on e.

M' halts on some strings $\Leftrightarrow$ M' halts on all strings
$\Leftrightarrow$ M halts on empty string

(d) The argument for part (c) works here well.

(e) We will reduce the problem in Part(d) to this one.
Given "M", our reduction constructs the string

$$\tau(\text{“}M\text{”}) = \text{“}M\text{”}\ \text{“}y\text{”}$$

where "y" is the description of the machine that immediately accepts any input.
The two machines M and y accept the same inputs iff M accepts all inputs.

(f) and (g) (ommited).

5.5 Properties of Recursive Languages

Theorem: L is recursive iff L and $\overline{L}$ are both r.e..

Proof:

- If L is recursive then L is r.e. (proved already).
- If L is recursive then $\overline{L}$ is recursive and hence $\overline{L}$ is r.e. (proved already).
- If L and $\overline{L}$ are both r.e. then here is a decision procedure for L:

> *Given $w \in \sum^*$, run semidecision procedures for L and $\overline{L}$ simultaneously (on alternate steps, for example).*
> *Since either $w \in L$ or $w \in \overline{L}$, one of these semidecision procedures will halt eventually.*

Definition: We say that a Turing machine M **enumerates** the language L iff for some fixed state q of M,

$$L = \{w : (s, \triangleright\sqcup) \vdash_M (q, \triangleright\sqcup w)\}.$$

A language is **Turing-enumerable** iff there is a Turing machine that enumerates it.

Theorem: A language is recursively enumerable iff it is Turing-enumerable.

Proof: $\Rightarrow$

Suppose TM M semidecides L. We want to construct a TM M' that enumerates L.

Basic idea:

M' should systematically

- generate strings over the alphabet of L, and
- run M on them, then
- put a string on the output tape if M would halt when given that string as input.

To make sure M' will discover all halting computations by M, dovetail all of the computations by M:

1. Do 1 step of M's computation on w_0
2. Do 2 steps of M on w_0 and w_1
3. Do 3 steps on each of w_0, w_1, w_2

.....

Here $w_0, w_1, \cdots$ is the lexicographic enumeration of $\sum^*$.

($\Leftarrow$) Conversely, suppose M enumerates L. We want to show that L is r.e..

Given w, run M on the blank tape. Every time M passes through state q (the "enumeration state") pause to see if w is on the output tape and halt if it is.

The language semidecided by this process is exactly the language enumerated by M.

Definition: Let M be a Turing machine M enumerating a language L. We say M **lexicographically enumerates** L if the following is true, where q is the special "display" state: Whenever $(q, \triangleright \sqcup w) \vdash_M^+ (q, \triangleright \sqcup w')$, then w' comes lexicographically after w. A language is **lexicographically Turing enumerable** iff there is a Turing machine that lexicographically enumerates it.

Theorem: A language is recursive iff it is lexicographically Turing-enumerable.

Proof: $\Rightarrow$

If L is recursive, then to enumerate L in order, generate all of $\sum^*$ in order and test each string for membership in L, enumerating those that are members.

$\Leftarrow$

to test if $w \in L$, enumerate L and wait till either w or a lexically later string is enumerated. ????

Rice Theorem

- An **index** is a string "M" for some TM M
- "M" is an **index** for the r.e. set $L(M)$

Example:

Set	Index (example)
H	"U"
$\emptyset$	"M" where M is any TM that has no transitions to the halt state

An index set is the set of all indices for some collection of r.e. sets, that is if $\mathscr{C}$ is a class of r.e. sets:
Let $\mathscr{I}(\mathscr{C}) = \{\text{“}M\text{”} : L(M) \in \mathscr{C}\}$

Example:

$\mathscr{C}$	$\mathscr{I}(\mathscr{C})$
$\{\emptyset\}$	The indices of all TM that halt on no input
$\{S : S$ is finite$\}$	The indices of all TMs that halt on only a finite number of input strings
$\{S \subseteq \sum^* : S$ is r.e. and $e \in S\}$	The indices of all TMs that halt on the empty string

Theorem: If S is a class of recursively enumerabale languages such that $\mathscr{I}(S)$ is neither empty nor the set of all indices, then $\mathscr{I}(S)$ is undecidable

---

i.e. almost all questions of the form:
"Here is a TM M; Does M halt on this kind of input?"
are undecidable.

e. g. Is $L(M) = \emptyset$ Is $L(M) =$ finite?
Is $L(M) = \sum^*$ Is $e \in L(M)$

Proof: Suppose $\mathscr{I}(S) \neq \emptyset$ and $\mathscr{I}(S)$ is not the set of all indices.

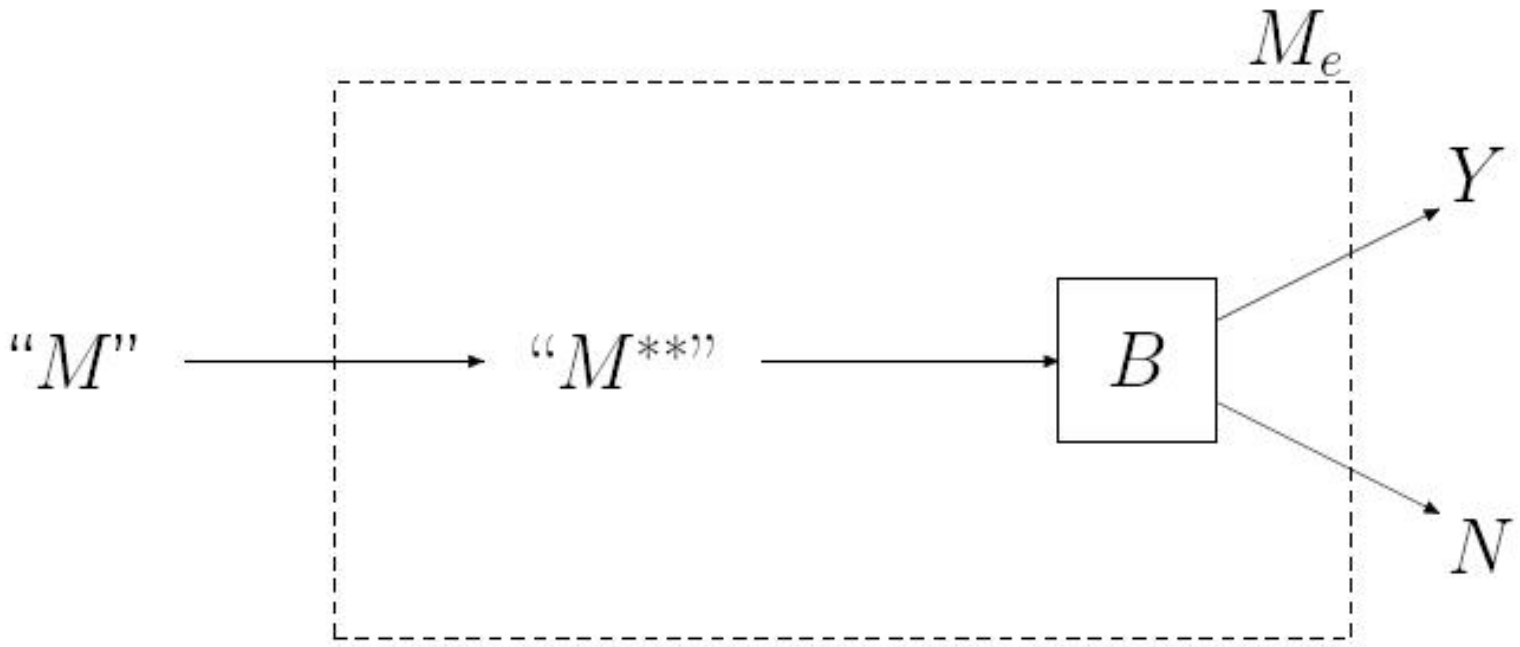
Suppose $\emptyset \notin S$ (Symmetrical argument works if $\emptyset \in S$) and "M^*" is any index in $\mathscr{I}(S)$.

Suppose $\mathscr{I}(S)$ were decidable. Then we could construct M_e deciding $\{\text{"}M\text{"}: M \text{ halts on } e\}$ as follows:
On input "M", construct"M^{**}" where M^{**} is TM that on input w

- first simulates M on input e
- then simulates M^* on input w (if M ever halts on e)

Then either $L(M^{**}) = \emptyset$ or $L(M^{**}) = L(M^*)$, depending on whether M halts on e.

So a decision procedure for $\mathscr{I}(S)$ provides one for $\{\text{“}M\text{”} : M \text{ halts on } e\}$.



In Picture Form

M_e : We are constructing this to decide
$\{$"M" : M halts on $e\}$

M^* : Some TM such that $L(M^*) \in S$
(exists since $\mathscr{I}(S)$ assumed nonempty)

B : "Black box" we assume decides $\mathscr{I}(S)$

M^{**} : TM whose encoding is constructed by M_e,
$L(M^{**}) = L(M^*)$ if $e \in L(M)$
$L(M^{**}) = \emptyset$ if $e \notin L(M)$

M_e halts yes on "M" iff B halts yes on "M^{**}"

iff $L(M^{**}) = L(M^*)$
iff $e \in L(M)$

5.6 More unsolvable problems

Grammar

Theorem: There is no algorithm to determine, given any grammar G and any string w, whether $w \in L(G)$.

Proof: Suppose there were such a decision procedure. Then we could use it to solve the halting problem:

> Given M and w, to determine if M halts on input w, construct a grammar G such that $L(M) = L(G)$ and determine if $w \in L(G)$.

Since the halting problem is unsolvable, so is this problem.

Remark:

There is a particular grammar G_0 for which this problem is unsolvable: namely, the grammar for the universal TM.

Theorem: Let G_1 and G_2 be two context-free grammars, to determine that whether $L(G_1) \cap L(G_2) = \emptyset$ is unsolvable.

Proof:

Reduce to this problem the problem of determining, given a string w and a general grammar G, whether $w \in L(G)$.

Show how to construct a computable function:

$$w; G \rightarrow G_1; G_2$$

such that G is a general grammar, $G_1; G_2$ CFGs, and $w \in L(G)$ iff $L(G_1) \cap L(G_2) = \emptyset$.

1. Let $G = (V; \sum; R; S)$. Let $\$ \notin V$. Then

$$L_1 = \{u_1\$v_1^R\$u_2\$v_2^R \cdots u_n\$v_n^R : n \geq 1,$$

$$u_i; v_i \in V^*, \text{ and } u_i \Rightarrow_G v_i \text{ for each } i\}$$

is context-free.

$L_1 = L(G_1)$ where G_1 has these rules:

$S \rightarrow S_1$

$S \rightarrow S_1\$S$

$S_1 \rightarrow aS_1a$ (for each $a \in V$)

$S_1 \rightarrow aS_2\beta^R$ (for each rule $\alpha \rightarrow \beta$ in R)

$S_2 \rightarrow aS_2a$ (for each $a \in V$)

$S_2 \rightarrow \$$

2.

$$L_2 = \{S\$u_1\$u_1^R\$\cdots\$u_n\$u_n^R\$w^R : n \geq 0; u_1; \cdots; u_n \in V^*\}$$

is context-free.

$L_2 = L(G_2)$ where

$S' \rightarrow S\$S''w^R$
$S'' \rightarrow S_1\$S'' \mid e$
$S_1 \rightarrow aS_1a$ (for each $a \in V$)
$S_1 \rightarrow \$$

3. $L_1 \cap L_2 = \emptyset$ iff $w \in L(G)$.

$$L_1 \cap L_2 = \{u_0\$u_1^R\$u_1\$u_2^R\$\cdots\$u_{n-1}^R\$u_{n-1}\$u_n^R :$$

$$u_0 = S$$

$$u_n = w$$

$$u_{i-1} \Rightarrow_G u_i,\ i = 0, \cdots, n-1$$

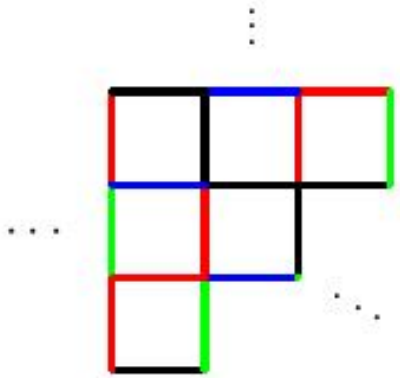
$$n \geq 1\}$$

That is, $L_1 \cap L_2$ encodes all legal derivations of w from S.

Tiling

Given a finite set of patterns for square tiles:

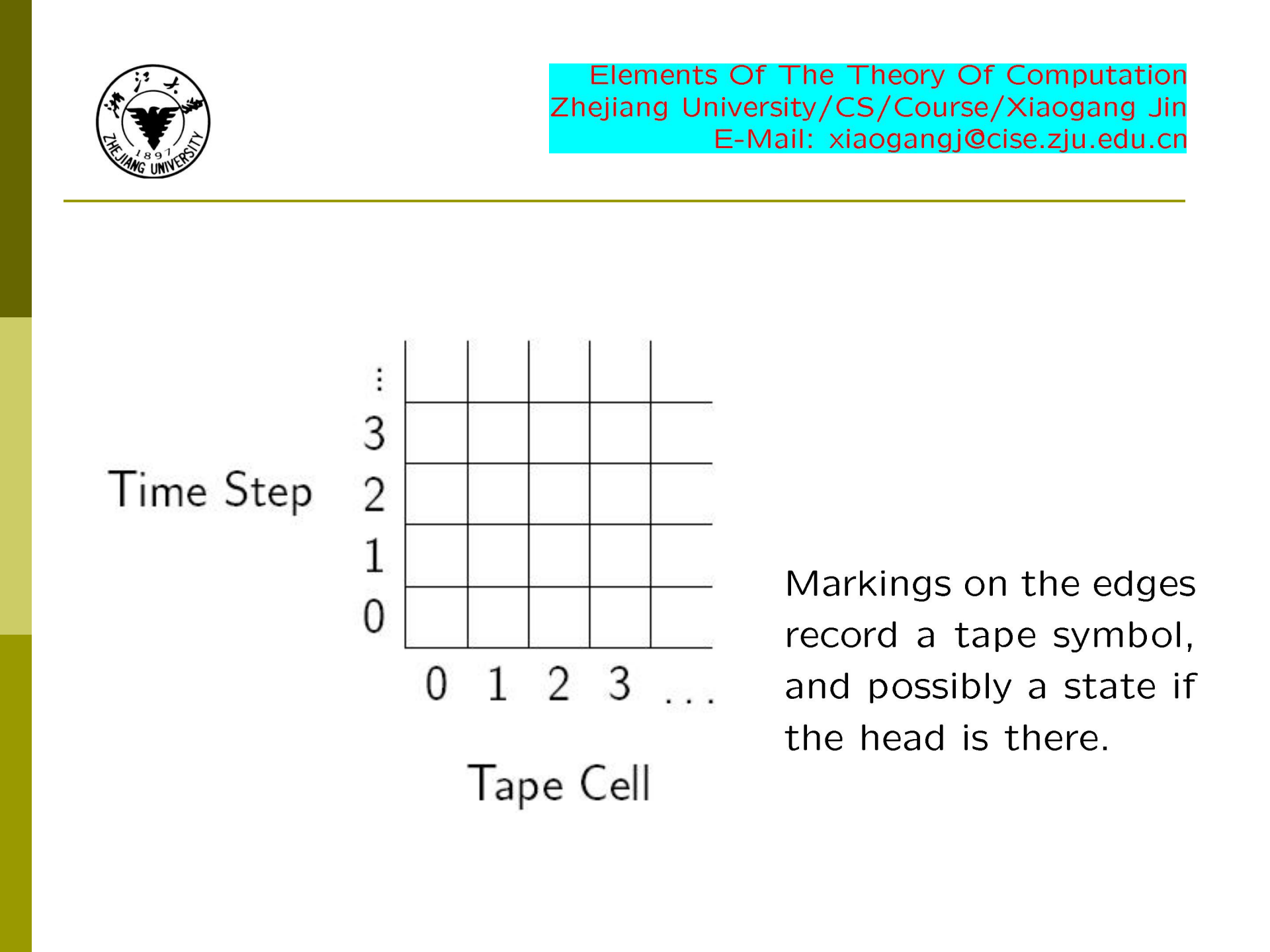
Is it possible to tile the whole plane with tiles of these patterns in such a way that the abutting edges match?



This "unconstrained" tiling problem is unsolvable though the proof is not easy.

If we are allowed to fix the tile at the origin and ask whether the first quadrant can be tiled, it is easier to show the undecidability.

Reduce TMs $\longmapsto^{T}$ sets of tiles so that:
M does not halt on $e \Leftrightarrow T(M)$ tiles the first quadrant.



Markings on the edges record a tape symbol, and possibly a state if the head is there.