Sec. 4: Turing Machines

There are two problems with the model of a URM:

- Execution of a single URM instruction might take arbitrarily long:
  - Consider $\text{succ}(n)$.
  - If $R^n$ contains in binary $111 \cdots 111$, this instruction replaces it by $1000 \cdots 000$.
  - We have to replace $k$ symbols 1 by 0.
  - $k$ is arbitrary
    - this single step might take arbitrarily long time.

First Problem of URMs

- That incrementing a number by one takes arbitrarily many steps happens on a real computer as well:
  - If we want to represent arbitrary big numbers on the computer, we have to represent them by multiple machine integers.
  - Then incrementing a number by one will correspond to arbitrarily many machine instructions (although usually only a few).
  - However, often in complexity theory this problem is ignored because the effect is marginal in real applications.
  - The exception are applications in which very big integers occur, e.g. tests for primality. There this effect cannot be ignored any more.

First Problem of URMs

- If one takes this effect into account, one needs in many examples to multiply the running time by a factor of $\ln(n)$, where $n$ is the largest number occurring.
  - Therefore URMs unsuitable as a basis for defining the precise complexity of algorithms.
  - However, there are theorems linking complexity of URMs to actual complexities of algorithms.

Second Problem of URMs

- We aim at a notion of computability, which covers all possible ways of computing something, independently of any concrete machine.
  - URMs are a model of computation which covers current standard computers.
  - However, there might be completely different notions of computability, based on symbolic manipulations of a sequence of characters, where it might be more complicated to see directly that all such computations can be simulated by a URM.
  - It is more easy to see that such notions are covered by the Turing machine model of computation.
Idea of a Turing Machine

- Idea of a Turing machine (TM): Analysis of a computation carried out by a human being (agent) on a piece of paper.

\[
\begin{array}{c}
\begin{array}{c}
15 \cdot 16 = \\
15 \\
90 \\
240
\end{array}
\end{array}
\]

Steps in this formulation:
- Algorithm should be deterministic.
  → The agent will use only finitely many symbols, put at discrete positions on the paper.

We can replace a two-dimensional piece of paper by one potentially infinite tape, by using a special symbol for a line break.
- Each entry on this tape is called a cell:

\[
\begin{array}{c}
\cdots \quad 1 \quad 5 \quad . \quad 1 \quad 6 \quad = \quad \text{CR} \quad \text{CR} \quad \text{CR} \quad 1 \quad 5 \quad \text{CR} \quad \cdots
\end{array}
\]

Steps in Formalising TMs
- In the real situation, agent can look at several cells at the same time, but bounded by his physical capability. Can be simulated by looking at one cell only at any time, and moving around in order to get information about neighbouring cells.
Steps in Formulising TMs

In the real situation, an agent can make arbitrary jumps, but bounded by the physical ability of the agent.
Each such jump can be simulated by finitely many one-step jumps.
→ Restriction to one-step movements.

... 1 5 . 1 6 = CR ... 1 5 CR ...

↑ Head
Steps in Formulising TMs

In the real situation, an agent can make arbitrary jumps, but bounded by the physical ability of the agent.
Each such jump can be simulated by finitely many one-step jumps.
→ Restriction to one-step movements.

... 1 5 . 1 6 = CR ... 1 5 CR ...

Agent operates purely mechanistically:
Reads a symbol, and depending on it changes it and makes a movement.
Agent himself will have only finite memory.
→ There is a finite state of the agent, and, depending on the state and the symbol at the head, a next state, a new symbol, and a movement is chosen.

... 1 5 . 1 6 = CR ... 1 5 CR ...

... 1 5 . 1 6 = CR ... 1 5 CR ...

... 1 5 . 0 6 = CR ... 1 5 CR ...

Agent operates purely mechanistically:
Reads a symbol, and depending on it changes it and makes a movement.
Agent himself will have only finite memory.
→ There is a finite state of the agent, and, depending on the state and the symbol at the head, a next state, a new symbol, and a movement is chosen.
Steps in Formulising TMs

Agent operates purely mechanistically:
Reads a symbol, and depending on it changes it and makes a movement.
Agent himself will have only finite memory.
→ There is a finite state of the agent, and, depending on the state and the symbol at the head, a next state, a new symbol, and a movement is chosen.

Definition of TMs

A Turing machine is a quintuple \((\Sigma, S, I, \downarrow, s_0)\), where
\(\Sigma\) is a finite set of symbols, called the alphabet of the Turing machine. On the tape, the symbols in \(\Sigma\) will be written.
\(S\) is a finite set of states.

\[
\begin{array}{c}
\cdots \ 1 \ 5 \ \ . \ 1 \ 6 \ = \ \ CR \ \ \ \ \ \ \ \ 1 \ 5 \ CR \ \cdots \\
\uparrow \ s_0
\end{array}
\]
Meaning of Instructions

- A instruction \((s, a, s', a', D) \in I\) means the following:
  - If the Turing machine is in state \(s\), and the symbol at position of the head is \(a\), then
  - the state is changed to \(s'\),
  - the symbol at this position is changed to \(a'\),
  - if \(D = L\), the head moves left,
  - if \(D = R\), the head moves right.

Example:
\[
(s_0, 1, s_1, 0, R) \\
(s_1, 0, s_2, 7, L)
\]
Meaning of Instructions

A instruction \((s, a, s', a', D) \in I\) means the following:

- If the Turing machine is in state \(s\), and the symbol at position of the head is \(a\), then
  - the state is changed to \(s'\),
  - the symbol at this position is changed to \(a'\),
  - if \(D = L\), the head moves left,
  - if \(D = R\), the head moves right.

Example:

\[
\begin{align*}
(s_0, 1, s_1, 0, R) \\
(s_1, 0, s_2, 7, L)
\end{align*}
\]

\[\cdots \quad 1 \quad 5 \quad . \quad 0 \quad 7 = \quad \text{CR} \quad \mid \quad \mid \quad \mid \quad 1 \quad 5 \quad \text{CR} \quad \cdots\]

\[\uparrow \]

\[s_2\]

Meaning of Instructions

- Note that for the above it is important that for every \(s \in S\), \(a \in \Sigma\) there is at most one \(s', a', D\) s.t. \((s, a, s', a', D) \in S\).
  - Without this condition, there might be more than one choice of selecting a new tape symbol, next state and direction.
  - If we omit this definition, we obtain a non-deterministic TM. In this case the machine selects in each step one of the possible choices (provided there exist one) at random.

- If the Turing machine is in a state \(s\) and reads symbol \(a\) at his head, and there are no \(s', a', D\) s.t. \((s, a, s', a', D) \in I\), then the Turing machine stops.

Visualisation of TMs

A TM

\((\Sigma, S, I, \bot, s_0)\)

can be visualised by a labelled graph as follows:

- Vertices: states (i.e. \(S\)).
- Edges: If \((s, a, t, b, D) \in I\), then there is an edge

\[
\begin{array}{ccc}
  s & \xrightarrow{a/b, D} & t \\
\end{array}
\]

- Furthermore we write an arrow to the initial state coming from nowhere.
- If there are several vertices from \(s\) to \(s'\), one draws only one arrow with one label for each vertex.

Example

The Turing machine with initial state \(s_0\) and instructions

\[
\begin{align*}
(s_0, 0, s_0, 0, R), \\
(s_0, 1, s_0, 0, R), \\
(s_0, \bot, s_1, \bot, L), \\
(s_1, 0, s_1, 0, L), \\
(s_1, \bot, s_2, \bot, R)
\end{align*}
\]

is visualised as follows (we write \(B\) instead of \(\bot\)):

\[
\begin{array}{ccc}
  s_0 & \xrightarrow{B/B, L} & s_1 \\
  s_1 & \xrightarrow{B/B, R} & s_2 \\
  0/0, R & \text{or} & 0/0, L \\
  1/0, R
\end{array}
\]
Example

- The TM on the previous slide sets the binary number the head is pointing to to zero, provided to the left of the head there are is a blank.

**Exercise:**
- This example assumes that the TM points to the left most digit of a binary number.
- Modify this TM, so that it works as well if the TM points initially to any digit of a binary number.

Example of a TM

- Development of a TM with \( \Sigma = \{0, 1, \_\_\} \), where \( \_\_ \) is the symbol for the blank entry.
- Functionality of the TM:
  - Assume initially the following:
    - The tape contains binary number,
    - The rest of the tape contains \( \_\_ \).
    - The head points to any digit of the number.
    - The TM in state \( s_0 \).
  - Then the TM stops after finitely many steps and then the tape contains the original number incremented by one,
    - the rest of tape contains \( \_\_ \),
    - the head points to most significant bit.
Construction of the TM

- TM is $\{\{0, 1, \_\_\_\}; S, I, \_\_\_\_; s_0\}$.
- States $S$ and instructions $I$ developed in the following.

Step 1

Initially, move head to least significant bit.
- I.e. as long as symbol at head is 0 or 1, move right, leave symbol as it is.
- If symbol is $\_\_\_\_$, move head left, leave symbol again as it is.

Achieved by the following instructions:

- $(s_0, 0, s_0, 0, R)$
- $(s_0, 1, s_0, 1, R)$
- $(s_0, \_\_\_\_, s_1, \_\_\_\_, L)$

At the end TM is in state $s_1$.

---

Step 1

Initially, move head to least significant bit.
- I.e. as long as symbol at head is 0 or 1, move right, leave symbol as it is.
- If symbol is $\_\_\_\_$, move head left, leave symbol again as it is.

Achieved by the following instructions:

- $(s_0, 0, s_0, 0, R)$
- $(s_0, 1, s_0, 1, R)$
- $(s_0, \_\_\_\_, s_1, \_\_\_\_, L)$

At the end TM is in state $s_1$. 

---
Step 1

- Initially, move head to least significant bit.
  - I.e. as long as symbol at head is 0 or 1, move right, leave symbol as it is.
  - If symbol is \( \_\_\_ \), move head left, leave symbol again as it is.

Achieved by the following instructions:

\[
(s_0, 0, s_0, 0, R) \\
(s_0, 1, s_0, 1, R) \\
(s_0, \_\_\_, s_1, \_\_\_, L)
\]

At the end TM is in state \( s_1 \).

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
\vdots & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & \ldots \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\uparrow \\
S_0
\end{array}
\]
Step 1

Initially, move head to least significant bit.
- I.e. as long as symbol at head is 0 or 1, move right, leave symbol as it is.
- If symbol is \( \perp, \) move head left, leave symbol again as it is.

Achieved by the following instructions:
\[
\begin{align*}
(s_0, 0, s_0, 0, \text{R}) \\
(s_0, 1, s_0, 1, \text{R}) \\
(s_0, \perp, s_1, \perp, \text{L})
\end{align*}
\]

At the end TM is in state \( s_1 \).

\[ \cdots \mid 1 \mid 1 \mid 1 \mid \cdots \]

\[ \uparrow \]

\[ s_0 \]

\[ s_1 \]
**Step 1**

- Initially, move head to least significant bit.
  - I.e. as long as symbol at head is 0 or 1, move right, leave symbol as it is.
  - If symbol is $\perp$, move head left, leave symbol again as it is.

- Achieved by the following instructions:
  
  $(s_0, 0, s_0, 0, R)$
  $(s_0, 1, s_0, 1, R)$
  $(s_0, \perp, s_1, \perp, L)$

- At the end TM is in state $s_1$.

---

**Step 2**

- Otherwise:
  - Then the representation of the number contains at the end one 0 followed by ones only.
  - Includes case where the least significant digit is 0.
  - Example 1: $b = (0100010111)_2$, one 0 followed by 3 ones.
  - Example 2: $b = (0100010010)_2$, least significant digit is 0.

- Let $b = (b_0b_1 \cdots b_k \underbrace{111\cdots111}_l)_2$.

- $b + 1$ obtained by replacing the final block of ones by 0 and the 0 by 1:
  
  $b + 1 = (b_0b_1 \cdots b_k1\underbrace{000\cdots000}_l)_2$.

---

**Step 2 – General Situation**

- We have to replace, as long as we find ones, the ones by zeros, and move left, until we encounter a 0 or a $\perp$, which is replaced by a 1.

- So we need a new state $s_2$, and the following instructions
  
  $(s_1, 1, s_1, 0, L)$
  $(s_1, 0, s_2, 1, L)$
  $(s_1, \perp, s_2, 1, L)$

- At the end the head will be one field to the left of the 1 written, and the state will be $s_2$. 

---

**Step 2**

- Increasing a binary number $b$ done as follows:
  - **Case number consists of 1 only:**
    - I.e. $b = (\underbrace{111\cdots111}_k)_2$.
    - $b + 1 = (1000\cdots000)_2$.
    - Obtained by replacing all ones by zeros and then replacing the first blank symbol by 1.
Step 2 – General Situation

We have to replace, as long as we find ones, the ones by zeros, and move left, until we encounter a 0 or a \( \_\_\_\_ \), which is replaced by a 1.

So we need a new state \( s_2 \), and the following instructions

\[
(s_1, 1, s_1, 0, L) \\
(s_1, 0, s_2, 1, L) \\
(s_1, \_\_\_, s_2, 1, L)
\]

At the end the head will be one field to the left of the 1 written, and the state will be \( s_2 \).

\[
\cdots 1 0 1 0 0 1 0 1 1 1 \quad \cdots
\]

\[\uparrow\]

\[s_1\]

\[
\text{Step 2 – General Situation}
\]

We have to replace, as long as we find ones, the ones by zeros, and move left, until we encounter a 0 or a \( \_\_\_\_ \), which is replaced by a 1.

So we need a new state \( s_2 \), and the following instructions

\[
(s_1, 1, s_1, 0, L) \\
(s_1, 0, s_2, 1, L) \\
(s_1, \_\_\_, s_2, 1, L)
\]

At the end the head will be one field to the left of the 1 written, and the state will be \( s_2 \).

\[
\cdots 1 0 1 0 0 1 0 1 0 0 \quad \cdots
\]

\[\uparrow\]

\[s_1\]
Step 2 – General Situation

We have to replace, as long as we find ones, the ones by zeros, and move left, until we encounter a 0 or a _., which is replaced by a 1.

So we need a new state $s_2$, and the following instructions

\[
\begin{align*}
(s_1, 1, s_0, 0, L) \\
(s_1, 0, s_2, 1, L) \\
(s_1, _., s_2, 1, L)
\end{align*}
\]

At the end the head will be one field to the left of the 1 written, and the state will be $s_2$.

At the end the head will be one field to the left of the 1 written, and the state will be $s_2$.

\[
\begin{array}{cccccccc}
\cdots & 1 & 0 & 1 & 0 & 0 & 1 & 0 & \cdots \\
\uparrow &   &   &   &   &   &   &   & \\
   & s_2 &   &   &   &   &   &   & \\
\end{array}
\]

Step 3

Finally, we have to move the most significant bit, which is done as follows

\[
\begin{align*}
(s_2, 0, s_2, 0, L) \\
(s_2, 1, s_2, 1, L) \\
(s_2, _., s_3, _., R)
\end{align*}
\]

The program terminates in state $s_3$.

\[
\begin{array}{cccccccc}
\cdots & 1 & 0 & 1 & 0 & 0 & 1 & 0 & \cdots \\
\uparrow &   &   &   &   &   &   &   & \\
   & s_2 &   &   &   &   &   &   & \\
\end{array}
\]
Step 3

Finally, we have to move the most significant bit, which is done as follows

\[(s_2, 0, s_2, 0, L)\]
\[(s_2, 1, s_2, 1, L)\]
\[(s_2, \_, \_, s_3, \_, R)\]

The program terminates in state \(s_3\).

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
\cdots & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots \\
\hline
s_2
\end{array}
\]

\[\uparrow\]

\[s_2\]
Finally, we have to move the most significant bit, which is done as follows:

\[(s_2, 0, s_2, 0, L)\]
\[(s_2, 1, s_2, 1, L)\]
\[(s_2, \_, s_3, \_, R)\]

The program terminates in state \(s_3\).
**Complete TM**

![Diagram of TM transitions]

**Notation: \( \text{bin} \)**

- TMs usually operate on binary numbers.
- Therefore we define for a natural number \( \text{bin}(n) \) as the sequence of digits representing the unique standard binary representation of \( n \).
- So \( \text{bin}(n) \) has no leading zeros, except for \( \text{bin}(0) := 0 \).
- Examples:
  - \( \text{bin}(0) = 0 \),
  - \( \text{bin}(1) = 1 \),
  - \( \text{bin}(2) = 10 \),
  - \( \text{bin}(3) = 11 \),
  - \( \text{bin}(4) = 100 \), etc.

**Notation: \( \tilde{\text{bin}} \)**

- In order to read off the final result, we need to interpret an arbitrary finite sequence of 0, 1 as a binary number, even if it has leading zeros.
- We define \( \tilde{\text{bin}}(n) \) as one of the possible binary representations of \( n \), allowing leading 0.
  - So \( \tilde{\text{bin}}(1) \) can be 1, 01, 001, etc.
  - In the special case 0 we treat the empty string as one of the possible representations, so \( \tilde{\text{bin}}(0) \) contains the empty string, 0, 00, 000, etc.

**Notation:**

- When carrying out intermediate calculations, it is easier to refer to \( \tilde{\text{bin}}(n) \) rather than \( \text{bin}(n) \).
  - E.g. we can set a number on the tape easily to \( \tilde{\text{bin}}(0) \) by overwriting it with 0s.
  - In order to set it to \( \text{bin}(0) \) one would need to make sure that exactly one 0 remains. Then one usually has to shift left the content of the tape to the right of the original number.
Definition 4.1

Let $T = (\Sigma, S, I, \downarrow, s_0)$ be a Turing machine with $\{0, 1\} \subseteq \Sigma$.

Define for every $k \in \mathbb{N}$ $T^{(k)} : \mathbb{N}^k \overset{\sim}{\rightarrow} \mathbb{N}$, where $T^{(k)}(a_0, \ldots, a_{k-1})$ is computed as follows:

- **Initialisation:**
  - The head is an arbitrary position.
  - Starting from this, we write $\text{bin}(a_0) \downarrow \text{bin}(a_1) \downarrow \cdots \downarrow \text{bin}(a_{k-1})$ on the tape.
  - E.g. if $k = 3$, $a_0 = 0$, $a_1 = 3$, $a_2 = 2$ then we write $0 \downarrow 11 \downarrow 10$.
  - All other cells contain $\downarrow \downarrow$.
  - The state is set to $s_0$.

- **Iteration:** Run the TM, until it stops.

**Output:**

- **Case 1:** The TM stops.
  - Only finitely many cells are non-blank.
  - Let tape, starting from the head-position, contain $b_0b_1 \cdots b_{k-1}c$ where $b_i \in \{0, 1\}$ and $c \not\in \{0, 1\}$.
  - (k might be 0).
  - Let $a = (b_0, \ldots, b_{k-1})_2$.
  - (in case $k = 0$, $a = 0$).
  - This means $b_0 \cdots b_{k-1}$ is one of the choices for $\text{bin}(a)$.
  - Then $T^{(k)}(a_0, \ldots, a_{k-1}) \simeq a$.

---

**Definition 4.1**

**Example:** Let $\Sigma = \{0, 1, a, b, \downarrow\}$.

- If the tape starting with the head is as follows:
  - $01010 \downarrow 0101 \downarrow$
  - or $01010 a b \downarrow$
  - output is $(01010)_2 = 10$.

- If tape starting with the head is as follows:
  - $a b \downarrow$
  - or $a$,
  - or $\downarrow \downarrow$,
  - the output is 0.

- **Case 2:** Otherwise.
  - Then $T^{(k)}(a_0, \ldots, a_{k-1}) \uparrow$.

---

**Definition 4.2**

$f : \mathbb{N}^k \overset{\sim}{\rightarrow} \mathbb{N}$ is **Turing-computable**, in short **TM-computable**, if $f = T^{(k)}$ for some TM $T$, the alphabet of which contains $\{0, 1\}$.

**Example:** That $\text{succ} : \mathbb{N} \overset{\sim}{\rightarrow} \mathbb{N}$ and $\text{zero} : \mathbb{N} \overset{\sim}{\rightarrow} \mathbb{N}$ are Turing-computable was shown above.
Remark

If the tape of the Turing machine initially contains only finitely many cells which are not blank, then at any step during the execution of the TM only finitely many cells are non blank.

Follows since in each step at most one cell can be modified to become non-blank.

So in finitely many steps only finitely many cells can be converted from blank to non-blank.

Theorem 4.3

If \( f : \mathbb{N}^n \rightarrow \mathbb{N} \) is URM-computable, then it is as well Turing-computable by a TM with alphabet \( \{0, 1, \_\} \).

Proof of Theorem 4.3

Assume

\[ f = \bigcup_{n} \]

\( U \) refers only to \( R_0, \ldots, R_{l-1} \) and \( l > n \).

We define a TM \( T \), which simulates \( U \). Done as follows:

That the registers \( R_0, \ldots, R_{l-1} \) contain \( a_0, \ldots, a_{l-1} \) is simulated by the tape containing \( \text{bin}(a_0)\_\ldots\_\text{bin}(a_{l-1}) \).

An instruction \( I_j \) will be simulated by states \( s_{j,0}, \ldots, s_{j,i} \) with instructions for those states.
Conditions on the Simulation

- Assume the URM $U$ is in a state s.t.
  - $R_0, \ldots, R_{l-1}$ contain $a_0, \ldots, a_{l-1}$,
  - the URM is about to execute $I_j$.
- Assume after executing $I_j$, the URM is in a state where
  - $R_0, \ldots, R_{l-1}$ contain $b_0, \ldots, b_{l-1}$,
  - the PC contains $k$.
- Then we want that, if configuration of the TM $T$ is, s.t.
  - the tape contains $\text{bin}(a_0) \ldots \text{bin}(a_{l-1})$,
  - and the TM is in state $s_{j,0}$,
- then the TM reaches a configuration s.t.
  - the tape contains $\text{bin}(b_0) \ldots \text{bin}(b_{l-1})$,
  - the TM is in state $s_{k,0}$.

Example

- Assume the URM is about to execute instruction
  - $I_4 = \text{pred}(2)$ (i.e. PC = 4),
  - with register contents
    \[
    \begin{array}{c|c|c}
    R_0 & R_1 & R_2 \\
    \hline
    2 & 1 & 3 \\
    \end{array}
    \]
- Then the URM will end with
  - PC = 5
  - and register contents
    \[
    \begin{array}{c|c|c}
    R_0 & R_1 & R_2 \\
    \hline
    2 & 1 & 2 \\
    \end{array}
    \]

Proof of Theorem 4.3

- Furthermore, we need initial states $s_{\text{init},0}, \ldots, s_{\text{init},j}$ and corresponding instructions, s.t.
  - if the TM initially contains
    \[
    \text{bin}(b_0) \ldots \text{bin}(b_{n-1})
    \]
  - it will reach state $s_{0,0}$ with the tape containing
    \[
    \text{bin}(b_0) \ldots \text{bin}(b_{l-1}) \text{bin}(0_{l-n} \cdot \text{bin}(b_{n-1}) \ldots 0_{l-n} \cdot \cdot \cdot \cdot 0_{l-n}) \]
    \text{l} - \text{n times}
Proof of Theorem 4.3

Assume the run of the URM, starting with $R_i$ containing $a_{0,i} = a_i$ for $i = 0, \ldots, n - 1$, and $a_{0,i} = 0$ for $i = n, \ldots, l - 1$ is as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>$R_0$</th>
<th>$R_1$</th>
<th>$\ldots$</th>
<th>$R_{n-1}$</th>
<th>$R_n$</th>
<th>$\ldots$</th>
<th>$R_{l-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_0$</td>
<td>$a_0$</td>
<td>$a_1$</td>
<td>$\ldots$</td>
<td>$a_{n-1}$</td>
<td>$0$</td>
<td>$\ldots$</td>
<td>$0$</td>
</tr>
<tr>
<td>$I_k$</td>
<td>$a_{0,0}$</td>
<td>$a_{0,1}$</td>
<td>$\ldots$</td>
<td>$a_{0,n-1}$</td>
<td>$a_{0,n}$</td>
<td>$\ldots$</td>
<td>$a_{0,l-1}$</td>
</tr>
<tr>
<td>$I_k$</td>
<td>$a_{1,0}$</td>
<td>$a_{1,1}$</td>
<td>$\ldots$</td>
<td>$a_{1,n-1}$</td>
<td>$a_{1,n}$</td>
<td>$\ldots$</td>
<td>$a_{1,l-1}$</td>
</tr>
<tr>
<td>$I_k$</td>
<td>$a_{2,0}$</td>
<td>$a_{2,1}$</td>
<td>$\ldots$</td>
<td>$a_{2,n-1}$</td>
<td>$a_{2,n}$</td>
<td>$\ldots$</td>
<td>$a_{2,l-1}$</td>
</tr>
</tbody>
</table>

Example

Consider the URM program $U$:

$I_0 = \text{ifzero}(0, 3)$
$I_1 = \text{pred}(0)$
$I_2 = \text{ifzero}(1, 0)$

$U^{(1)}(a) \simeq 0$. 

Proof of Theorem 4.3

Then the corresponding TM will successively reach the following configurations:

<table>
<thead>
<tr>
<th>State</th>
<th>Tape contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_{\text{init.}}$</td>
<td>$\text{bin}(a_0) _ \text{bin}(a_1) _ \ldots _ \text{bin}(a_{n-1}) _ \ldots _ \text{bin}(a_{l-1})$</td>
</tr>
<tr>
<td>$s_0,0$</td>
<td>$\text{bin}(a_0) _ \text{bin}(a_1) _ \ldots _ \text{bin}(a_{n-1}) _ \text{bin}(0) _ \ldots _ \text{bin}(0)$</td>
</tr>
<tr>
<td>$s_{k_0,0}$</td>
<td>$\text{bin}(a_{0,0}) _ \text{bin}(a_{0,1}) _ \ldots _ \text{bin}(a_{0,l-1})$</td>
</tr>
<tr>
<td>$s_{k_1,0}$</td>
<td>$\text{bin}(a_{1,0}) _ \text{bin}(a_{1,1}) _ \ldots _ \text{bin}(a_{1,l-1})$</td>
</tr>
<tr>
<td>$s_{k_2,0}$</td>
<td>$\text{bin}(a_{2,0}) _ \text{bin}(a_{2,1}) _ \ldots _ \text{bin}(a_{2,l-1})$</td>
</tr>
</tbody>
</table>

Example

$I_0 = \text{ifzero}(0, 3)$
$I_1 = \text{pred}(0)$
$I_2 = \text{ifzero}(1, 0)$
Example

$I_0 = \text{ifzero}(0,3)$
$I_1 = \text{pred}(0)$
$I_2 = \text{ifzero}(1,0)$

A run of $U^{(1)}(2)$ is as follows:

Instruction $R_0$ $R_1$

Example

$I_0 = \text{ifzero}(0,3)$
$I_1 = \text{pred}(0)$
$I_2 = \text{ifzero}(1,0)$

A run of $U^{(1)}(2)$ is as follows:

Instruction $R_0$ $R_1$

$R_0$ $R_1$

Example

$I_0 = \text{ifzero}(0,3)$
$I_1 = \text{pred}(0)$
$I_2 = \text{ifzero}(1,0)$

A run of $U^{(1)}(2)$ is as follows:

Instruction $R_0$ $R_1$

$R_0$ $R_1$

Example

$I_0 = \text{ifzero}(0,3)$
$I_1 = \text{pred}(0)$
$I_2 = \text{ifzero}(1,0)$

A run of $U^{(1)}(2)$ is as follows:

Instruction $R_0$ $R_1$

$R_0$ $R_1$

$R_0$ $R_1$
Example

$I_0 = \text{ifzero}(0,3)$
$I_1 = \text{pred}(0)$
$I_2 = \text{ifzero}(1,0)$

A run of $U^{(1)}(2)$ is as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>$R_0$</th>
<th>$R_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_0$</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$I_1$</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$I_2$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$I_0$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**Example**

\[ I_0 = \text{ifzero}(0,3) \]
\[ I_1 = \text{pred}(0) \]
\[ I_2 = \text{ifzero}(1,0) \]

A run of \( U^{(1)}(2) \) is as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R₀</th>
<th>R₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₀</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>I₁</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>I₂</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I₀</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I₁</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I₂</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I₀</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Example**

\[ I_0 = \text{ifzero}(0,3) \]
\[ I_1 = \text{pred}(0) \]
\[ I_2 = \text{ifzero}(1,0) \]

A run of \( U^{(1)}(2) \) is as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R₀</th>
<th>R₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₀</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>I₁</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>I₂</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I₀</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I₁</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>I₂</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I₀</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Corresponding TM Simulation**

\[ I_0 = \text{ifzero}(0,3) \]
\[ I_1 = \text{pred}(0) \]
\[ I_2 = \text{ifzero}(1,0) \]

A run of \( U^{(1)}(2) \) is as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R₀</th>
<th>R₁</th>
<th>State of TM</th>
<th>Content of Tape</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₀</td>
<td>2</td>
<td>0</td>
<td>( s_{\text{init},0} )</td>
<td>( \text{bin}(0) )</td>
</tr>
</tbody>
</table>
| I₁          | 2  | 0  | \( s_{0,0} \) | \( \text{bin}(2) \)
| I₂          | 1  | 0  | \( s_{1,0} \) | \( \text{bin}(2) \)
| I₀          | 1  | 0  | \( s_{2,0} \) | \( \text{bin}(1) \)
| I₁          | 1  | 0  | \( s_{0,0} \) | \( \text{bin}(1) \)
| I₂          | 0  | 0  | \( s_{1,0} \) | \( \text{bin}(0) \)
| I₀          | 0  | 0  | \( s_{2,0} \) | \( \text{bin}(0) \)
| I₃          | 0  | 0  | \( s_{3,0} \) | \( \text{bin}(0) \)
Proof of Theorem 4.3

If we have defined this we have

If

\[ U^n(a_0, \ldots, a_{n-1}) \downarrow , \]
\[ U^n(a_0, \ldots, a_{n-1}) \simeq c , \]

then \( U \) eventually stops with \( R \), containing some values \( b_i \), where \( b_0 = c \).

Then, the TM \( T \) starting with

\[ \text{bin}(a_0) \ldots \text{bin}(a_{n-1}) \]

will eventually terminate in a configuration

\[ \text{bin}(b_0) \ldots \text{bin}(b_{k-1}) . \]

for some \( k \geq n \).
Therefore \( T^n(a_0, \ldots, a_{n-1}) \simeq b_0 = c. \)

Proof of Theorem 4.3

It follows

\[ U^{(n)} = T^{(n)} , \]

and the proof is complete, if the simulation has been introduced.

The following slides contain a detailed proof, which will not be presented in the lecture this year.

Jump over remaining proof.

Proof of Theorem 4.3

Informal description of the simulation of URM instructions.

- **Initialisation.**
  Initially, the tape contains \( \text{bin}(a_0) \ldots \text{bin}(a_{n-1}) \).
  We need to obtain configuration:
  \[ \text{bin}(a_0) \ldots \text{bin}(a_{n-1}) \underbrace{\text{bin}(0) \ldots \text{bin}(0)}_{l - n \text{ times}} . \]

  Achieved by
  - moving head to the end of the initial configuration
  - inserting, starting from the next blank, \( l - n \)-times \( 0 \),
  - then moving back to the beginning.
Proof of Theorem 4.3

Simulation of URM instructions.

Simulation of instruction $I_k = \text{succ}(j)$.
Need to increase $(j+1)^{\text{st}}$ binary number by 1. Initial configuration:

\[
\begin{align*}
\text{bin}(c_0) & | \text{bin}(c_1) | \cdots | \text{bin}(c_j) | \cdots | \text{bin}(c_l) \\
\uparrow
\end{align*}
\]

$s_{k,0}$

First move to the $(j+1)^{\text{st}}$ blank to the right. Then we are at the end of the $(j+1)^{\text{st}}$ binary number.

\[
\begin{align*}
\text{bin}(c_0) & | \text{bin}(c_1) | \cdots | \text{bin}(c_j) | \cdots | \text{bin}(c_l) \\
\uparrow
\end{align*}
\]

Simulation of instruction $I_k = \text{pred}(j)$.
Assume the configuration at the beginning is:

\[
\begin{align*}
\text{bin}(c_0) & | \text{bin}(c_1) | \cdots | \text{bin}(c_j) | \cdots | \text{bin}(c_l) \\
\uparrow
\end{align*}
\]

Now perform the operation for increasing by 1 as above.
At the end we obtain:

\[
\begin{align*}
\text{bin}(c_0) & | \text{bin}(c_1) | \cdots | \text{bin}(c_j+1) | \cdots | \text{bin}(c_l) \\
\uparrow
\end{align*}
\]

It might be that we needed to write over the separating blank a 1, in which case we have:

\[
\begin{align*}
\text{bin}(c_0) & | \text{bin}(c_1) | \cdots | \text{bin}(c_{j-1}) | \text{bin}(c_j+1) | \cdots | \text{bin}(c_l) \\
\uparrow
\end{align*}
\]
Proof of Theorem 4.3

Initially: \( \text{bin}(c_0) \ldots \text{bin}(c_j) \ldots \text{bin}(c_l) \)

Finally: \( \text{bin}(c_0) \ldots \text{bin}(c_j \dashv 1) \ldots \text{bin}(c_l) \)

Move to end of the \((j + 1)\)st number.

Check, if the number consists only of zeros or not.

If it consists only of zeros, \(\text{pred}(j)\) doesn’t change anything.

Otherwise, number is of the form \(b_0 \ldots b_k 1 \underbrace{00 \ldots 0}_{l’ \text{ times}}\).

Replace it by \(b_0 \ldots b_k 1 1 \underbrace{11 \ldots 1}_{l’ \text{ times}}\).

Done as for \(\text{succ}\).

Proof of Theorem 4.3

We have achieved

\( \text{bin}(c_0) \ldots \text{bin}(c_1) \ldots \text{bin}(c_j \dashv 1) \ldots \text{bin}(c_l) \)

Move back to the beginning:

\( \text{bin}(c_0) \ldots \text{bin}(c_1) \ldots \text{bin}(c_j \dashv 1) \ldots \text{bin}(c_l) \)

Remark

We will later show that all TM-computable functions are URM-computable.

This will be done by showing that all TM-computable functions are partial recursive, all partial recursive functions are URM-computable.

This will be easier than showing directly that TM-computable functions are URM-computable.

Therefore the set of TM-computable functions and the set of URM-computable functions coincide.
Extension to Arbitrary Alphabets

- Let $A$ be a finite alphabet s.t. \( \cup \notin A \), and $B := A^*$.  
- To a Turing machine $T = (\Sigma, S, I, \cup, s_0)$ with $A \subseteq \Sigma$ corresponds a partial function $T^{(A,n)} : B^n \rightarrow B$, where $T^{(A,n)}(a_0, \ldots, a_{n-1})$ is computed as follows:  
  - Initially write $a_0 \cup \cdots \cup a_{n-1}$ on the tape, otherwise \( \bot \).  
  - Start in state $s_0$ on the left most position of $a_0$.  
  - Iterate TM as before.  
  - In case of termination, the output of the function is $c_0 \cdots c_{l-1}$, if the tape contains, starting with the head position $c_0 \cdots c_{l-1}d$ with $c_i \in A, d \notin A$.  
  - Otherwise, the function value is undefined.

Turing-Computable Predicates

- A predicate $A$ is Turing-decidable, iff $\chi_A$ is Turing-computable.  
- Instead of simulating $\chi_A$  
  - means to write the output of $\chi_A$ (a binary number 0 or 1) on the tape  
  - it is more convenient, to take TM with two additional special states $s_{\text{true}}$ and $s_{\text{false}}$ corresponding to truth and falsity of the predicate.

Extension to Arbitrary Alphabets

- Notion is modulo encoding of $A^*$ into $\mathbb{N}$ equivalent to the notion of Turing-computability on $\mathbb{N}$.  
- However, when considering complexity bounds, this notation might be more appropriate.  
- Avoids encoding/decoding into $\mathbb{N}$.

Turing-Computable Predicates

- Then a predicate is Turing decidable, if, when we write initially the inputs as before on the tape and start executing the TM,  
  - it always terminates in $s_{\text{true}}$ or $s_{\text{false}}$,  
  - and it terminates in $s_{\text{true}}$, iff the predicate holds for the inputs,  
  - and in $s_{\text{false}}$, otherwise.  
- The latter notion is equivalent to the first notion.  
- Usually the latter one is taken as basis for complexity considerations.