Closing the circle
Overview

1. Encode configurations of TM’s.
2. Encode TM’s themselves.
3. Define a primitive recursive function

$$conf(m, x, t)$$

that yields the configuration reached by the TM with code $m$ on input $x$ after time $t$.

4. Use $conf(m, x, t)$ to define the recursive function computed by the TM.
To encode the tape, we use

- a **left number**, which results from interpreting the tape left of the scanned square as a binary numeral, prefixed by infinitely many superfluous 0’s;

- a **right number**, which results from interpreting the rest of the tape, consisting of the scanned square and the portion to its right, as a binary numeral **written backwards**.
For the sake of presentation, suppose that the TM takes only one argument, $x$.

Then the initial tape has one block of $x + 1$ strokes and is otherwise blank, and the leftmost stroke is scanned.

So the left number is 0, and the right number is

$$2^0 + 2^1 + 2^2 + \cdots + 2^{x-1} + 2^x = 2^{x+1} - 1.$$

We define a primitive recursive function

$$start(x) = 2^{x+1} - 1.$$
Computing the scanned symbol

Let $r$ be the right number. The scanned symbol is

- 0 if the binary representation of $r$ ends with 0, i.e. if $r$ is even.
- 1 if the binary representation of $r$ ends with 1, i.e. if $r$ is odd.

So the scanned symbol is the remainder of dividing $r$ by 2:

$$\text{scan}(r) = \text{rem}(r, 2).$$

As seen earlier, $\text{rem}$ is primitive recursive; so the same is true for $\text{scan}$. 
Writing a 0

Suppose the action is $W_0$.

- The left number remains the same.
- If the scanned square already contains 0, the right number remains the same; otherwise, it is decreased by 1.
- Letting $p$ be the left number and $r$ the right number, we have

$$newleft_0(p, r) = p$$
$$newright_0(p, r) = r - \text{scan}(r).$$
Writing a $1$

In a similar way, we get a primitive recursive functions for writing a $1$:

$$newleft_1(p, r) = p$$

$$newright_1(p, r) = r + 1 - \text{scan}(r).$$
Moving left: new left number

Let $p$ be the pre-move left number, and let $p^*$ be the post-move left number.

The binary representation of $p^*$ is obtained by chopping of the last 0 or 1.

This means that $p^*$ is $p$ divided by 2 (and rounded down), so $p^*$ is given by

$$newleft_L(p, r) = quo(p, 2).$$
Moving left: new right number

- Let $r$ be the pre-move right number, and let $r^*$ be the post-move right number.
- Let $p_0$ be the symbol to the left of the scanned square.
- The binary representation of $r^*$ is obtained from the one for $r$ by appending $p_0$, so
  \[ r^* = 2r + p_0. \]
- We have $p_0 = \text{rem}(p, 2)$; so $r^*$ is given by
  \[ \text{newright}_L(p, r) = 2r + \text{rem}(p, 2). \]
Moving right

By reversing the rôles of $p$ and $r$, we get the functions for moving right:

\[
newleft_R(p, r) = 2p + \text{rem}(r, 2)
\]
\[
newright_R(p, r) = \text{quo}(r, 2).
\]
Before we proceed, we encode the actions as follows:

<table>
<thead>
<tr>
<th>action</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_0$</td>
<td>0</td>
</tr>
<tr>
<td>$W_1$</td>
<td>1</td>
</tr>
<tr>
<td>$L$</td>
<td>2</td>
</tr>
<tr>
<td>$R$</td>
<td>3</td>
</tr>
</tbody>
</table>
The action as an extra argument

We can now define new versions of \textit{newleft} and \textit{newright} that take the action as an extra argument:

\[
\text{newleft}(p, r, a) = \begin{cases} 
  p & \text{if } a = 0 \text{ or } a = 1 \\
  \text{quo}(p, 2) & \text{if } a = 2 \\
  2p + \text{rem}(r, 2) & \text{if } a = 3 
\end{cases}
\]

This is a \textbf{definition by cases}, so \textit{newleft} is primitive recursive. Similarly for \textit{newright}. 

Encoding configurations

- A configuration consists of a tape and a state.
- So a configuration can be represented as a triple \((p, q, r)\), where is \(p\) and \(r\) are left and right numbers, and \(q\) is a state.
- We can use the primitive recursive encoding
  \[ c = \text{triple}(p, q, r) = 2^p \cdot 3^q \cdot 5^r \]
  and its primitive recursive decodings
  \[
  \begin{align*}
  \text{left} &= \text{lo}(c, 2) \\
  \text{state} &= \text{lo}(c, 3) \\
  \text{right} &= \text{lo}(c, 5).
  \end{align*}
  \]
Extracting the final value

- Suppose that the TM halts in a standard final configuration \( c = \text{triple}(p, q, r) \).

- If the result is \( y \), then there is a single block with \( y + 1 \) strokes, which are the binary representation of \( r \); so

\[
    r = 2^{y+1} - 1.
\]

- So \( y = \text{lo}(r + 1, 2) - 1 \), i.e. \( y \) is given by the primitive recursive function

\[
    \text{value}(c) = \text{lo}(\text{right}(c) + 1, 2) - 1.
\]
In a standard final configuration $c = \text{triple}(p, q, r)$, we have $p = 0$, and the previous slide implies that

$$\exists y < r. r = 2^{y+1} - 1.$$ 

So $c$ represents a s.f.c. iff the relation $\text{is}_\text{std}(c)$ is defined as

$$\text{is}_\text{std}(c) \iff \text{left}(c) = 0 \text{ and } \exists y < \text{right}(c). \text{right}(c) = 2^{y+1} - 1$$

holds. Because the $\exists$ is bounded, this relation is primitive recursive.
Encoding TM’s

We have seen an encoding of TM’s before; now we use an improved version. Recall that a TM can be presented by a transition table, e.g.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$W_1q_1$</td>
<td>$Lq_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$W_1q_2$</td>
<td>$Lq_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$W_1q_3$</td>
<td></td>
</tr>
</tbody>
</table>

We use the convention that $q_1$ is the starting state.
Encoding TM’s

By introducing a halting state $q_0$, we can assume that the transition table is defined everywhere. E.g. the table from the previous slide becomes

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$W_0 q_0$</td>
<td>$W_1 q_0$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$W_1 q_1$</td>
<td>$L q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$W_1 q_2$</td>
<td>$L q_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$W_1 q_3$</td>
<td>$W_1 q_0$</td>
</tr>
</tbody>
</table>

The table can be written as a list, e.g. $(W_0, q_0, W_1, q_0, W_1, q_1, L, q_2, W_1, q_2, L, q_3, W_1, q_3, W_1, q_0)$. 
Encoding TM’s

The entries of the list can be represented by natural numbers, e.g.

\[(W_0, q_0, W_1, q_0, W_1, q_1, L, q_2, W_1, q_2, L, q_3, W_1, q_3, W_1, q_0)\]

becomes

\[(0, 0, 1, 0, 1, 1, 2, 2, 1, 2, 2, 3, 1, 3, 1, 0).\]

This list can be encoded into a natural number \(m\) which is the code of the TM, e.g.

\[2^0 \cdot 3^0 \cdot 5^1 \cdot 7^0 \cdot 11^1 \cdot 13^1 \cdot 17^2 \ldots .\]
Using the encoding

\[(W_0, q_0, W_1, q_0, W_1, q_1, L, q_2, W_1, q_2, L, q_3, W_1, q_3, W_1, q_0)(0, 0, 1, 0, 1, 1, 2, 2, 1, 2, 2, 3, 1, 3, 1, 0)\]

- The action when scanning symbol \(i\) in state \(q\) is given by entry number \(4q + 2i\).
- The next state is given by entry number \(4q + 2i + 1\).
- We have primitive recursive functions

\[\begin{align*}
\text{action}(m, q, r) &= \text{entry}(m, 4q + 2 \cdot \text{scan}(r)) \\
\text{newstate}(m, q, r) &= \text{entry}(m, 4q + 2 \cdot \text{scan}(r) + 1).
\end{align*}\]
Next, we define a primitive recursive function \( \text{conf}(m, x, t) \) that returns the configuration reached by TM with code \( m \) on input \( x \) after \( t \) steps.

- After 0 steps we have

\[
\text{conf}(m, x, 0) = \text{triple}(0, 1, \text{start}(x)).
\]

- We define

\[
\text{conf}(m, x, t + 1) = \text{newconf}(m, \text{conf}(m, x, t)).
\]
Defining $\text{newconf}(m, c)$

1. Apply $\text{left}$, $\text{state}$, and $\text{right}$ to $c$ to obtain the left number $p$, the number $q$ of the state, and the right number $r$.

2. Apply $\text{action}$ and $\text{newstate}$ to $(m, q, r)$ to obtain the number $a$ of the action, and the number $q^*$ of the new state.

3. Let $\text{newconf}(m, c) =$
   \[ \text{triple}(\text{newleft}(p, r, a), q^*, \text{newright}(p, r, a)) \].

We used only composition, so $\text{newconf}$ is primitive recursive.
Halting in standard configuration

The TM is halted when \( \text{state}(\text{conf}(m, x, t)) = 0 \).

So, letting

\[
\text{stdh}(m, x, t) = \begin{cases} 
0 & \text{if } \text{state}(\text{conf}(m, x, t)) = 0 \\
\text{and } \text{is_std}(\text{conf}(m, x, t)) & \\
1 & \text{otherwise,}
\end{cases}
\]

the machine is halted in a standard configuration iff \( \text{stdh}(m, x, t) = 0 \).

This is a definition by cases, so the function \( \text{stdh} \) is primitive recursive.
The time of halting

The time (if any) when the machine halts in a standard configuration is

\[
\text{halt}(m, x) = \begin{cases} 
\text{the least } t & \text{if such a } t \\
\text{such that} & \text{exists} \\
\text{stdh}(m, x, t) = 0 \\
\text{undefined} & \text{otherwise.}
\end{cases}
\]

The function \textit{halt} is recursive, because it is defined by minimization over a (primitive) recursive function (\textit{stdh}).
Putting it all together

Let $F(m, x) = \text{value}(\text{conf}(m, x, \text{halt}(m, x)))$.

$F(m, x)$ is the value of the function computed by the TM with code $m$ for argument $x$.

$F$ is recursive, because it is defined by composition from recursive functions.

Let $f(x) = F(m, x)$.

$f$ is the function computed by the TM with code $m$, and $f$ is recursive.
So we have proved:

**Theorem.** Every Turing-computable function is recursive.

This closes the circle.