Recursive functions and Turing machines

Madhavan Mukund, **S P Suresh**

Programming Language Concepts
Lecture 18, 20 March 2025

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]
  - Equivalent to Turing machines

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]
  - Equivalent to Turing machines
- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **composition** from $g : \mathbb{N}^l \to \mathbb{N}$ and $h_1, \ldots, h_l : \mathbb{N}^k \to \mathbb{N}$ if

$$f(\vec{n}) = g(h_1(\vec{n}), \ldots, h_l(\vec{n}))$$

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]
  - Equivalent to Turing machines
- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **composition** from $g : \mathbb{N}^l \to \mathbb{N}$ and $h_1, \ldots, h_l : \mathbb{N}^k \to \mathbb{N}$ if

$$f(\vec{n}) = g(h_1(\vec{n}), \ldots, h_l(\vec{n}))$$

- **Notation**: $f = g \circ (h_1, h_2, \ldots, h_l)$

Recursive functions

- Recursive functions [**Dedekind**, **Skolem**, **Gödel**, **Kleene**]
  - Equivalent to Turing machines
- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **composition** from $g : \mathbb{N}^l \to \mathbb{N}$ and $h_1, \ldots, h_l : \mathbb{N}^k \to \mathbb{N}$ if

$$f(\vec{n}) = g(h_1(\vec{n}), \ldots, h_l(\vec{n}))$$

- **Notation**: $f = g \circ (h_1, h_2, \ldots, h_l)$
- Simulated by a sequence of assignments

Recursive functions

- $f : \mathbb{N}^{k+1} \to \mathbb{N}$ is obtained by **primitive recursion** from $g : \mathbb{N}^k \to \mathbb{N}$ and $h : \mathbb{N}^{k+2} \to \mathbb{N}$ if

$$f\ (0,\ \vec{n}) = g(\vec{n})$$
$$f\ (i+1,\ \vec{n}) = h(i, f\ (i,\ \vec{n}),\ \vec{n})$$

Recursive functions

- $f : \mathbb{N}^{k+1} \to \mathbb{N}$ is obtained by **primitive recursion** from $g : \mathbb{N}^k \to \mathbb{N}$ and $h : \mathbb{N}^{k+2} \to \mathbb{N}$ if

$$f\ (0,\ \vec{n}) = g(\vec{n})$$
$$f\ (i+1,\ \vec{n}) = h(i, f\ (i,\ \vec{n}),\ \vec{n})$$

- **Note:** If g and h are total functions, so is f

Recursive functions

- $f : \mathbb{N}^{k+1} \to \mathbb{N}$ is obtained by **primitive recursion** from $g : \mathbb{N}^k \to \mathbb{N}$ and $h : \mathbb{N}^{k+2} \to \mathbb{N}$ if

$$f\ (0, \vec{n}) = g(\vec{n})$$
$$f\ (i+1, \vec{n}) = h(i, f\ (i, \vec{n}), \vec{n})$$

- **Note:** If g and h are total functions, so is f
- Equivalent to a `for` loop:

```
result = g(n1, ... , nk);        // f(0, n1, ... , nk)
for (i = 0; i < n; i++) {        // computing f(i+1, n1, ... , nk)
    result = h(i, result, n1, ... , nk);
}
return result;
```

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i, \vec{n}) = 0 \text{ and } \forall j < i : g(j, \vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i, \vec{n}) = 0 \text{ and } \forall j < i : g(j, \vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- **Notation**: $f(\vec{n}) = \mu i \, (g(i, \vec{n}) = 0)$

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i, \vec{n}) = 0 \text{ and } \forall j < i : g(j, \vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- **Notation**: $f(\vec{n}) = \mu i \, (g(i, \vec{n}) = 0)$
- f need not be total even if g is

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i, \vec{n}) = 0 \text{ and } \forall j < i : g(j, \vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- **Notation**: $f(\vec{n}) = \mu i \, (g(i, \vec{n}) = 0)$
- f need not be total even if g is
- If $f(\vec{n}) = i$, then $g(j, \vec{n})$ is defined for all $j \leqslant i$

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i, \vec{n}) = 0 \text{ and } \forall j < i : g(j, \vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Recursive functions

- $f : \mathbb{N}^k \to \mathbb{N}$ is obtained by **μ-recursion** or **minimization** from $g : \mathbb{N}^{k+1} \to \mathbb{N}$ if

$$f(\vec{n}) = \begin{cases} i & \text{if } g(i, \vec{n}) = 0 \text{ and } \forall j < i : g(j, \vec{n}) > 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- Equivalent to a `while` loop:

```
i = 0;
while (g(i, n1, ..., nk) > 0) {
    i = i + 1;
}
return i;
```

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

     Successor $S(n) = n + 1$

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

Zero	$Z(n) = 0$
Successor	$S(n) = n + 1$
Projection	$\Pi_i^k(n_1, \ldots, n_k) = n_i$

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

     Successor $S(n) = n + 1$

     Projection $\Pi_i^k(n_1, \ldots, n_k) = n_i$
  2. closed under composition and primitive recursion

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

     Successor $S(n) = n + 1$

     Projection $\Pi_i^k(n_1, \ldots, n_k) = n_i$
  2. closed under composition and primitive recursion
- The class of **(partial) recursive functions** is the smallest class of functions

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

     Successor $S(n) = n + 1$

     Projection $\Pi_i^k(n_1, \ldots, n_k) = n_i$
  2. closed under composition and primitive recursion
- The class of **(partial) recursive functions** is the smallest class of functions
  1. containing the initial functions

Recursive functions

- The class of **primitive recursive functions** is the smallest class of functions
  1. containing the **initial functions**

     Zero $Z(n) = 0$

     Successor $S(n) = n + 1$

     Projection $\Pi_i^k(n_1, \ldots, n_k) = n_i$
  2. closed under composition and primitive recursion
- The class of **(partial) recursive functions** is the smallest class of functions
  1. containing the initial functions
  2. closed under composition, primitive recursion and minimization

Primitive recursive functions: Examples

- $f\ (n) = n + 2$ is $S \circ S$

Primitive recursive functions: Examples

- $f\ (n) = n + 2$ is $S \circ S$
- $plus(n, m) = n + m$ is got by primitive recursion from $g = \Pi^1_1$ and $h = S \circ \Pi^3_2$

$$
\begin{array}{lcll}
plus(0, m) & = & g(m) & = \Pi^1_1(m) \\
 & & & = m \\
plus(n+1, m) & = & h(n, plus(n, m), m) & \\
 & = & (S \circ \Pi^3_2)(n, plus(n, m), m) & = S(plus(n, m)) \\
 & & & = (n + m) + 1 \\
 & & & = (n + 1) + m
\end{array}
$$

Primitive recursive functions: Examples

- $mult(n, m) = nm$ is got by primitive recursion from $g = Z$ and $h = plus \circ (\Pi^3_2, \Pi^3_3)$

$$
\begin{aligned}
mult(0, m) &= g(m) &&= Z(m) \\
&&&= 0 \\
mult(n+1, m) &= h(n, mult(n, m), m) \\
&= (plus \circ (\Pi^3_2, \Pi^3_3))(n, mult(n, m), m) \\
&&&= nm + m \\
&&&= (n+1)m
\end{aligned}
$$

Primitive recursive functions: Examples

- $exp(n, m) = m^n$ is got by primitive recursion from $g = S \circ Z$ and $h = mult \circ (\Pi^3_2, \Pi^3_3)$

$$
\begin{aligned}
exp(0, m) &= g(m) &&= (S \circ Z)(m) \\
&&&= 1 \\
exp(n+1, m) &= h(n, exp(n, m), m) \\
&= (mult \circ (\Pi^3_2, \Pi^3_3))(n, exp(n, m), m) \\
&&&= m^n \cdot m \\
&&&= m^{n+1}
\end{aligned}
$$

Primitive recursive functions: Examples

- Define $pred(n) = \begin{cases} 0 & \text{if } n = 0 \\ n - 1 & \text{otherwise} \end{cases}$

Primitive recursive functions: Examples

- Define $pred(n) = \begin{cases} 0 & \text{if } n = 0 \\ n-1 & \text{otherwise} \end{cases}$
- $pred = f \circ (\Pi^1_1, \Pi^1_1)$, where f is got by primitive recursion from $g = Z$ and $h = \Pi^3_1$

$$\begin{array}{lllll} f(0,m) & = & g(m) & = Z(m) & = 0 \\ f(n+1,m) & = & h(n, f(n,m), m) & = \Pi^3_1(n, f(n,m), m) & = n \end{array}$$

$$\begin{array}{lllll} pred(0) & = & (f \circ (\Pi^1_1, \Pi^1_1))(0) & = f(0,0) & = 0 \\ pred(n+1) & = & (f \circ (\Pi^1_1, \Pi^1_1))(n+1) & = f(n+1, n+1) & = n \end{array}$$

Primitive recursive functions: Examples

- Define $m \dot{-} n = \begin{cases} 0 & \text{if } m \leqslant n \\ m - n & \text{otherwise} \end{cases}$

Primitive recursive functions: Examples

- Define $m \dot{-} n = \begin{cases} 0 & \text{if } m \leqslant n \\ m - n & \text{otherwise} \end{cases}$
- $m \dot{-} n = f\ (n, m)$ where f is got by primitive recursion from $g = \Pi^1_1$ and $h = pred \cdot \Pi^3_2$

$$
\begin{array}{lllll}
f\ (0, m) & = & g(m) & = & \Pi^1_1(m) \\
 & & & = & m \qquad = \ m \dot{-} 0 \\
f\ (n+1, m) & = & h(n, f\ (n, m), m) & = & pred(\Pi^3_2(n, f\ (n, m), m)) \\
 & & & = & pred(m \dot{-} n) \qquad = \ m \dot{-} (n+1)
\end{array}
$$

Primitive recursive functions: Examples

- Define $m \dot{-} n = \begin{cases} 0 & \text{if } m \leqslant n \\ m - n & \text{otherwise} \end{cases}$
- $m \dot{-} n = f\ (n, m)$ where f is got by primitive recursion from $g = \Pi^1_1$ and $h = pred \cdot \Pi^3_2$

$$
\begin{array}{lllll}
f\ (0, m) & = g(m) & = \Pi^1_1(m) & & \\
 & & = m & = m \dot{-} 0 & \\
f\ (n+1, m) & = h(n, f\ (n, m), m) & = pred(\Pi^3_2(n, f\ (n, m), m)) & & \\
 & & = pred(m \dot{-} n) & = m \dot{-} (n+1) &
\end{array}
$$

- **Note the recursion on the second argument!**

Primitive recursive functions: Examples

- **Factorial**

$0! = 1$ $\quad$ $(n+1)! = (n+1) \cdot n!$

Primitive recursive functions: Examples

- **Factorial**

$$0! = 1 \qquad (n+1)! = (n+1) \cdot n!$$

- **Bounded sum** $g(z, \vec{x}) = \sum_{y \leqslant z} f(y, \vec{x})$:

$$g(0, \vec{x}) = f(0, \vec{x})$$
$$g(y+1, \vec{x}) = g(y, \vec{x}) + f(y+1, \vec{x})$$

Primitive recursive functions: Examples

- **Factorial**

$$0! = 1 \qquad (n+1)! = (n+1) \cdot n!$$

- **Bounded sum** $g(z, \vec{x}) = \sum_{y \leqslant z} f(y, \vec{x})$:

$$g(0, \vec{x}) = f(0, \vec{x})$$
$$g(y+1, \vec{x}) = g(y, \vec{x}) + f(y+1, \vec{x})$$

- **Bounded product** $g(z, \vec{x}) = \prod_{y \leqslant z} f(y, \vec{x})$:

$$g(0, \vec{x}) = f(0, \vec{x})$$
$$g(y+1, \vec{x}) = g(y, \vec{x}) \cdot f(y+1, \vec{x})$$

Primitive recursive functions and predicates

Definition

A relation $R \subseteq \mathbb{N}^k$ is primitive recursive if its characteristic function c_R is primitive recursive

Primitive recursive functions and predicates

Definition

A relation $R \subseteq \mathbb{N}^k$ is primitive recursive if its characteristic function c_R is primitive recursive

- c_{iszero} (and hence *iszero*) is primitive recursive:

$$iszero(0) = true \qquad c_{iszero}(0) = succ(Z(0))$$
$$iszero(n+1) = false \qquad c_{iszero}(n+1) = Z(n)$$

Primitive recursive functions and predicates

Definition

A relation $R \subseteq \mathbb{N}^k$ is primitive recursive if its characteristic function c_R is primitive recursive

- c_{iszero} (and hence *iszero*) is primitive recursive:

$$iszero(0) = true \qquad c_{iszero}(0) = succ(Z(0))$$
$$iszero(n+1) = false \qquad c_{iszero}(n+1) = Z(n)$$

- $\leqslant$ is primitive recursive: $x \leqslant y$ iff $iszero(x - y)$, so $c_{\leqslant}(x, y) = c_{iszero}(x - y)$

Primitive recursive functions and predicates

Definition

A relation $R \subseteq \mathbb{N}^k$ is primitive recursive if its characteristic function c_R is primitive recursive

- c_{iszero} (and hence *iszero*) is primitive recursive:

$$iszero(0) = true \qquad c_{iszero}(0) = succ(Z(0))$$
$$iszero(n+1) = false \qquad c_{iszero}(n+1) = Z(n)$$

- $\leqslant$ is primitive recursive: $x \leqslant y$ iff $iszero(x - y)$, so $c_{\leqslant}(x, y) = c_{iszero}(x - y)$
- Primitive recursive relations are closed under boolean operations: $c_{\neg\varphi} = 1 - c_{\varphi}$, $c_{\varphi \wedge \psi} = c_{\varphi} \cdot c_{\psi}$

Primitive recursive predicates: Examples

- Primitive recursive relations are closed under bounded universal quantification:

$$\text{If } \varphi(z, \vec{x}) = (\forall y \leqslant z)\psi(y, \vec{x}), \qquad \text{then } c_\varphi(z, \vec{x}) = \prod_{y \leqslant z} c_\psi(y, \vec{x})$$

Primitive recursive predicates: Examples

- Primitive recursive relations are closed under bounded universal quantification:

$$\text{If } \varphi(z, \vec{x}) = (\forall y \leqslant z)\psi(y, \vec{x}), \qquad \text{then } c_\varphi(z, \vec{x}) = \prod_{y \leqslant z} c_\psi(y, \vec{x})$$

- $x = y$, $x < y$, $\varphi \vee \psi$, $\varphi \to \psi$, $(\exists y \leqslant z)\varphi(y, \vec{x})$ etc. are primitive recursive, when φ and ψ are!

Primitive recursive predicates: Examples

- Primitive recursive relations are closed under bounded universal quantification:

$$\text{If } \varphi(z, \vec{x}) = (\forall y \leqslant z)\psi(y, \vec{x}), \qquad \text{then } c_\varphi(z, \vec{x}) = \prod_{y \leqslant z} c_\psi(y, \vec{x})$$

- $x = y$, $x < y$, $\varphi \vee \psi$, $\varphi \to \psi$, $(\exists y \leqslant z)\varphi(y, \vec{x})$ etc. are primitive recursive, when φ and ψ are!
- Closed under **bounded μ-recursion**:

$$\chi(z, \vec{x}) = \mu y_{\leqslant z}\, \varphi(y, \vec{x}) = \begin{cases} \mu y.\varphi(y, \vec{x}) & \text{if } (\exists y \leqslant z)\varphi(y, \vec{x}) \\ z + 1 & \text{otherwise} \end{cases}$$

Primitive recursive predicates: Examples

- Primitive recursive relations are closed under bounded universal quantification:

$$\text{If } \varphi(z, \vec{x}) = (\forall y \leqslant z)\psi(y, \vec{x}), \qquad \text{then } c_\varphi(z, \vec{x}) = \prod_{y \leqslant z} c_\psi(y, \vec{x})$$

- $x = y$, $x < y$, $\varphi \vee \psi$, $\varphi \to \psi$, $(\exists y \leqslant z)\varphi(y, \vec{x})$ etc. are primitive recursive, when φ and ψ are!
- Closed under **bounded μ-recursion**:

$$\chi(z, \vec{x}) = \mu y_{\leqslant z}\, \varphi(y, \vec{x}) = \begin{cases} \mu y.\varphi(y, \vec{x}) & \text{if } (\exists y \leqslant z)\varphi(y, \vec{x}) \\ z + 1 & \text{otherwise} \end{cases}$$

  - $\psi'(y, \vec{x}) := (\forall w < y)\neg\varphi(w, \vec{x})$

Primitive recursive predicates: Examples

- Primitive recursive relations are closed under bounded universal quantification:

$$\text{If } \varphi(z, \vec{x}) = (\forall y \leqslant z)\psi(y, \vec{x}), \qquad \text{then } c_\varphi(z, \vec{x}) = \prod_{y \leqslant z} c_\psi(y, \vec{x})$$

- $x = y$, $x < y$, $\varphi \vee \psi$, $\varphi \to \psi$, $(\exists y \leqslant z)\varphi(y, \vec{x})$ etc. are primitive recursive, when φ and ψ are!
- Closed under **bounded μ-recursion**:

$$\chi(z, \vec{x}) = \mu y_{\leqslant z}\, \varphi(y, \vec{x}) = \begin{cases} \mu y.\varphi(y, \vec{x}) & \text{if } (\exists y \leqslant z)\varphi(y, \vec{x}) \\ z + 1 & \text{otherwise} \end{cases}$$

  - $\psi'(y, \vec{x}) := (\forall w < y)\neg\varphi(w, \vec{x})$
  - $\psi(y, \vec{x}) := \varphi(y, \vec{x}) \wedge \psi'(y, \vec{x})$

Primitive recursive predicates: Examples

- Primitive recursive relations are closed under bounded universal quantification:

$$\text{If } \varphi(z, \vec{x}) = (\forall y \leqslant z)\psi(y, \vec{x}), \qquad \text{then } c_\varphi(z, \vec{x}) = \prod_{y \leqslant z} c_\psi(y, \vec{x})$$

- $x = y$, $x < y$, $\varphi \vee \psi$, $\varphi \to \psi$, $(\exists y \leqslant z)\varphi(y, \vec{x})$ etc. are primitive recursive, when φ and ψ are!
- Closed under **bounded μ-recursion**:

$$\chi(z, \vec{x}) = \mu y_{\leqslant z}\, \varphi(y, \vec{x}) = \begin{cases} \mu y.\varphi(y, \vec{x}) & \text{if } (\exists y \leqslant z)\varphi(y, \vec{x}) \\ z + 1 & \text{otherwise} \end{cases}$$

  - $\psi'(y, \vec{x}) := (\forall w < y)\neg\varphi(w, \vec{x})$
  - $\psi(y, \vec{x}) := \varphi(y, \vec{x}) \wedge \psi'(y, \vec{x})$
  - $\chi(z, \vec{x}) := \left(\sum_{y \leqslant z} y \cdot c_\psi(y, \vec{x})\right) + (z + 1) \cdot c_{\psi'}(z + 1, \vec{x})$

More primitive recursive predicates and functions

- x **divides** y

$$x|y \text{ iff } (\exists z \leqslant y)\ (x \cdot z = y)$$

More primitive recursive predicates and functions

- x **divides** y

$$x|y \text{ iff } (\exists z \leqslant y)\ (x \cdot z = y)$$

- x **is even**

$$even(x) \text{ iff } 2|x$$

More primitive recursive predicates and functions

- x **divides** y

$$x|y \text{ iff } (\exists z \leqslant y)\ (x \cdot z = y)$$

- x **is even**

$$even(x) \text{ iff } 2|x$$

- x **is odd**

$$odd(x) \text{ iff } \neg even(x)$$

More primitive recursive predicates and functions

- x **divides** y

$$x|y \text{ iff } (\exists z \leqslant y)\ (x \cdot z = y)$$

- x **is even**

$$even(x) \text{ iff } 2|x$$

- x **is odd**

$$odd(x) \text{ iff } \neg even(x)$$

- **rightmost a.k.a least significant bit of the binary representation of** x

$$lsb(x) := c_{odd}(x)$$

More primitive recursive predicates and functions

- x **divides** y

$$x|y \text{ iff } (\exists z \leqslant y)\,(x \cdot z = y)$$

- x **is even**

$$even(x) \text{ iff } 2|x$$

- x **is odd**

$$odd(x) \text{ iff } \neg even(x)$$

- **rightmost a.k.a least significant bit of the binary representation of** x

$$lsb(x) := c_{odd}(x)$$

- x **is a prime**

$$prime(x) \text{ iff } x \geqslant 2 \wedge (\forall y \leqslant x)(y|x \rightarrow y = 1 \vee y = x)$$

More primitive recursive functions

- **the n-th prime**

$$
\begin{aligned}
Pr(0) &= 2 \\
Pr(n+1) &= \text{the smallest prime greater than } Pr(n) \\
&= \mu y_{\leqslant Pr(n)!+1}\ (prime(y) \wedge y > Pr(n))
\end{aligned}
$$

More primitive recursive functions

- **the n-th prime**

$$\begin{aligned} Pr(0) &= 2 \\ Pr(n+1) &= \text{the smallest prime greater than } Pr(n) \\ &= \mu y_{\leqslant Pr(n)!+1}\ (prime(y) \wedge y > Pr(n)) \end{aligned}$$

- **the exponent of (the prime) k in the decomposition of y**

$$exp(y,k) = \mu x_{\leqslant y}\left[k^x | y \wedge \neg(k^{x+1}|y)\right]$$

Primitive recursive coding of the plane

- $\frac{x}{2} = \mu y_{\leqslant x}(2y \geqslant x)$

Primitive recursive coding of the plane

- $\frac{x}{2} = \mu y_{\leqslant x}(2y \geqslant x)$
- The **Cantor bijection** between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$ is primitive recursive:

$$pair(x, y) = \frac{(x + y)^2 + 3x + y}{2}$$

Primitive recursive coding of the plane

- $\frac{x}{2} = \mu y_{\leqslant x}(2y \geqslant x)$
- The **Cantor bijection** between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$ is primitive recursive:

$$pair(x, y) = \frac{(x + y)^2 + 3x + y}{2}$$

- The inverses are also primitive recursive:

$$fst(z) = \mu x_{\leqslant z}\left[(\exists y \leqslant z)(z = pair(x, y))\right]$$

$$snd(z) = \mu y_{\leqslant z}\left[(\exists x \leqslant z)(z = pair(x, y))\right]$$

Primitive recursive coding of sequences

- The sequence $x_1, \ldots, x_n$ (of length n) is coded by

$$Pr(0)^n \cdot Pr(1)^{x_1} \cdot Pr(2)^{x_2} \cdots Pr(n)^{x_n}$$

Primitive recursive coding of sequences

- The sequence $x_1, \ldots, x_n$ (of length n) is coded by

$$Pr(0)^n \cdot Pr(1)^{x_1} \cdot Pr(2)^{x_2} \cdots Pr(n)^{x_n}$$

- **n-th element of the sequence coded by x**

$$(x)_n = exp(x, Pr(n))$$

Primitive recursive coding of sequences

- The sequence $x_1, \ldots, x_n$ (of length n) is coded by

$$Pr(0)^n \cdot Pr(1)^{x_1} \cdot Pr(2)^{x_2} \cdots Pr(n)^{x_n}$$

- **n-th element of the sequence coded by x**

$$(x)_n = exp(x, Pr(n))$$

- **length of sequence coded by x**

$$ln(x) = (x)_0$$

Primitive recursive coding of sequences

- The sequence $x_1, \ldots, x_n$ (of length n) is coded by

$$Pr(0)^n \cdot Pr(1)^{x_1} \cdot Pr(2)^{x_2} \cdots Pr(n)^{x_n}$$

- **n-th element of the sequence coded by x**

$$(x)_n = exp(x, Pr(n))$$

- **length of sequence coded by x**

$$ln(x) = (x)_0$$

- **x codes a sequence**

$$Seq(x) \text{ iff } (\forall n \leqslant x)\,[(x)_n \neq 0 \longrightarrow n \leqslant ln(x)]$$

Turing machines

- $M = (Q, \Delta)$

Turing machines

- $M = (Q, \Delta)$
  - $Q = \{q_0, \ldots, q_l\}$ is a finite set

Turing machines

- $M = (Q, \Delta)$
  - $Q = \{q_0, \ldots, q_l\}$ is a finite set
  - q_0 is the **initial state**

Turing machines

- $M = (Q, \Delta)$
  - $Q = \{q_0, \ldots, q_l\}$ is a finite set
  - q_0 is the **initial state**
  - q_1 is the **final state**

Turing machines

- $M = (Q, \Delta)$
  - $Q = \{q_0, \ldots, q_l\}$ is a finite set
  - q_0 is the **initial state**
  - q_1 is the **final state**
  - $\Delta \subseteq Q \times \{0, 1\} \times \{0, 1, L, R\} \times Q$

Turing machines

- $M = (Q, \Delta)$
  - $Q = \{q_0, \ldots, q_l\}$ is a finite set
  - q_0 is the **initial state**
  - q_1 is the **final state**
  - $\Delta \subseteq Q \times \{0,1\} \times \{0,1,L,R\} \times Q$
  - For each $i \leqslant l$ and $a \in \{0,1\}$, there is at most one transition of the form $(q_i, a, \cdot, \cdot)$ **deterministic**

Turing machines

- $M = (Q, \Delta)$
  - $Q = \{q_0, \ldots, q_l\}$ is a finite set
  - q_0 is the **initial state**
  - q_1 is the **final state**
  - $\Delta \subseteq Q \times \{0,1\} \times \{0,1,L,R\} \times Q$
  - For each $i \leqslant l$ and $a \in \{0,1\}$, there is at most one transition of the form $(q_i, a, \cdot, \cdot)$ **deterministic**
  - There is no transition of the form $(q_1, \cdot, \cdot, \cdot) \in \Delta$ **machine stops on hitting the final state**

Turing machines

- $M = (Q, \Delta)$
  - $Q = \{q_0, \ldots, q_l\}$ is a finite set
  - q_0 is the **initial state**
  - q_1 is the **final state**
  - $\Delta \subseteq Q \times \{0,1\} \times \{0,1,L,R\} \times Q$
  - For each $i \leqslant l$ and $a \in \{0,1\}$, there is at most one transition of the form $(q_i, a, \cdot, \cdot)$ **deterministic**
  - There is no transition of the form $(q_1, \cdot, \cdot, \cdot) \in \Delta$ **machine stops on hitting the final state**
- **Two-way infinite tape** with tape cells indexed by $\mathbb{Z}$

Turing machines

- $M = (Q, \Delta)$
  - $Q = \{q_0, \ldots, q_l\}$ is a finite set
  - q_0 is the **initial state**
  - q_1 is the **final state**
  - $\Delta \subseteq Q \times \{0,1\} \times \{0,1,L,R\} \times Q$
  - For each $i \leqslant l$ and $a \in \{0,1\}$, there is at most one transition of the form $(q_i, a, \cdot, \cdot)$ **deterministic**
  - There is no transition of the form $(q_1, \cdot, \cdot, \cdot) \in \Delta$ **machine stops on hitting the final state**
- **Two-way infinite tape** with tape cells indexed by $\mathbb{Z}$
- One tape-head **always scanning the cell numbered** 0

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;
  - $t : \mathbb{Z} \to \{0, 1\}$ is the **tape content**;

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;
  - $t : \mathbb{Z} \to \{0, 1\}$ is the **tape content**;
  - $t(i) = 1$ for only finitely many $i \in \mathbb{Z}$.

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;
  - $t : \mathbb{Z} \to \{0, 1\}$ is the **tape content**;
  - $t(i) = 1$ for only finitely many $i \in \mathbb{Z}$.
- The tape content can be coded as two numbers:

$$L(C) := \sum_{j \leqslant 0} t(j) \cdot 2^{-j} \qquad R(C) := \sum_{j > 0} t(j) \cdot 2^{j-1}$$

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;
  - $t : \mathbb{Z} \to \{0, 1\}$ is the **tape content**;
  - $t(i) = 1$ for only finitely many $i \in \mathbb{Z}$.
- The tape content can be coded as two numbers:

$$L(C) := \sum_{j \leqslant 0} t(j) \cdot 2^{-j} \qquad R(C) := \sum_{j > 0} t(j) \cdot 2^{j-1}$$

  - $L(C)$ is the tape content to the left of (and including) cell 0, **from left to right**

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;
  - $t : \mathbb{Z} \to \{0, 1\}$ is the **tape content**;
  - $t(i) = 1$ for only finitely many $i \in \mathbb{Z}$.
- The tape content can be coded as two numbers:

$$L(C) := \sum_{j \leqslant 0} t(j) \cdot 2^{-j} \qquad R(C) := \sum_{j > 0} t(j) \cdot 2^{j-1}$$

  - $L(C)$ is the tape content to the left of (and including) cell 0, **from left to right**
  - $R(C)$ is the tape content to the right of (and excluding) cell 0, **from right to left**

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;
  - $t : \mathbb{Z} \to \{0, 1\}$ is the **tape content**;
  - $t(i) = 1$ for only finitely many $i \in \mathbb{Z}$.
- The tape content can be coded as two numbers:

$$L(C) := \sum_{j \leqslant 0} t(j) \cdot 2^{-j} \qquad R(C) := \sum_{j > 0} t(j) \cdot 2^{j-1}$$

  - $L(C)$ is the tape content to the left of (and including) cell 0, **from left to right**
  - $R(C)$ is the tape content to the right of (and excluding) cell 0, **from right to left**
  - Both sums are **effectively finite**, since $t(i) = 1$ for finitely many i

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;
  - $t : \mathbb{Z} \to \{0, 1\}$ is the **tape content**;
  - $t(i) = 1$ for only finitely many $i \in \mathbb{Z}$.
- The tape content can be coded as two numbers:

$$L(C) := \sum_{j \leqslant 0} t(j) \cdot 2^{-j} \qquad R(C) := \sum_{j > 0} t(j) \cdot 2^{j-1}$$

  - $L(C)$ is the tape content to the left of (and including) cell 0, **from left to right**
  - $R(C)$ is the tape content to the right of (and excluding) cell 0, **from right to left**
  - Both sums are **effectively finite**, since $t(i) = 1$ for finitely many i
- **Initial configuration** – (q_0, t) with $t(i) = 0$ for $i > 0$

Turing machine configurations

- Let $M = (Q, \Delta)$ be a Turing machine
- **Configuration** is a pair $C = (q, t)$ where:
  - $q \in Q$ is the **current state**;
  - $t : \mathbb{Z} \to \{0, 1\}$ is the **tape content**;
  - $t(i) = 1$ for only finitely many $i \in \mathbb{Z}$.
- The tape content can be coded as two numbers:

$$L(C) := \sum_{j \leqslant 0} t(j) \cdot 2^{-j} \qquad R(C) := \sum_{j > 0} t(j) \cdot 2^{j-1}$$

  - $L(C)$ is the tape content to the left of (and including) cell 0, **from left to right**
  - $R(C)$ is the tape content to the right of (and excluding) cell 0, **from right to left**
  - Both sums are **effectively finite**, since $t(i) = 1$ for finitely many i
- **Initial configuration** – (q_0, t) with $t(i) = 0$ for $i > 0$
- **Final configuration** – (q_1, t) with $t(i) = 0$ for $i > 0$

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M
- $C \xrightarrow{\delta} C'$ iff one of the following holds:

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M
- $C \xrightarrow{\delta} C'$ iff one of the following holds:
  - $\delta = (r, a, b, s)$, $b \in \{0, 1\}$, $q = r$, $q' = s$, $t(0) = a$, $t'(0) = b$ and $t'(j) = t(j)$ for all $j \neq 0$

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M
- $C \xrightarrow{\delta} C'$ iff one of the following holds:
  - $\delta = (r, a, b, s)$, $b \in \{0, 1\}$, $q = r$, $q' = s$, $t(0) = a$, $t'(0) = b$ and $t'(j) = t(j)$ for all $j \neq 0$
    - **replace a with b**

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M
- $C \xrightarrow{\delta} C'$ iff one of the following holds:
  - $\delta = (r, a, b, s)$, $b \in \{0, 1\}$, $q = r$, $q' = s$, $t(0) = a$, $t'(0) = b$ and $t'(j) = t(j)$ for all $j \neq 0$
    - **replace a with b**
  - $\delta = (r, a, L, s)$, $q = r$, $q' = s$, $t(0) = a$, and $t'(j) = t(j-1)$ for all j

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M
- $C \xrightarrow{\delta} C'$ iff one of the following holds:
  - $\delta = (r, a, b, s)$, $b \in \{0, 1\}$, $q = r$, $q' = s$, $t(0) = a$, $t'(0) = b$ and $t'(j) = t(j)$ for all $j \neq 0$
    - **replace a with b**
  - $\delta = (r, a, L, s)$, $q = r$, $q' = s$, $t(0) = a$, and $t'(j) = t(j-1)$ for all j
    - **Move the head one cell to the left and readjust indices** -1 is the new 0, and $j-1$ the new j

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M
- $C \xrightarrow{\delta} C'$ iff one of the following holds:
  - $\delta = (r, a, b, s)$, $b \in \{0, 1\}$, $q = r$, $q' = s$, $t(0) = a$, $t'(0) = b$ and $t'(j) = t(j)$ for all $j \neq 0$
    - **replace a with b**
  - $\delta = (r, a, L, s)$, $q = r$, $q' = s$, $t(0) = a$, and $t'(j) = t(j-1)$ for all j
    - **Move the head one cell to the left and readjust indices** -1 is the new 0, and $j - 1$ the new j
  - $\delta = (r, a, R, s)$, $q = r$, $q' = s$, $t(0) = a$, and $t'(j) = t(j+1)$ for all j.

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M
- $C \xrightarrow{\delta} C'$ iff one of the following holds:
  - $\delta = (r, a, b, s)$, $b \in \{0, 1\}$, $q = r$, $q' = s$, $t(0) = a$, $t'(0) = b$ and $t'(j) = t(j)$ for all $j \neq 0$
    - **replace a with b**
  - $\delta = (r, a, L, s)$, $q = r$, $q' = s$, $t(0) = a$, and $t'(j) = t(j-1)$ for all j
    - **Move the head one cell to the left and readjust indices** -1 is the new 0, and $j-1$ the new j
  - $\delta = (r, a, R, s)$, $q = r$, $q' = s$, $t(0) = a$, and $t'(j) = t(j+1)$ for all j.
    - **Move the head one cell to the right and readjust indices** 1 is the new 0, and $j+1$ the new j

Transition between configurations

- Let $M = (Q, \Delta)$ be a Turing machine and $\delta \in \Delta$
- Let $C = (q, t)$ and $C' = (q', t')$ be configurations of M
- $C \xrightarrow{\delta} C'$ iff one of the following holds:
  - $\delta = (r, a, b, s)$, $b \in \{0, 1\}$, $q = r$, $q' = s$, $t(0) = a$, $t'(0) = b$ and $t'(j) = t(j)$ for all $j \neq 0$
    - **replace a with b**
  - $\delta = (r, a, L, s)$, $q = r$, $q' = s$, $t(0) = a$, and $t'(j) = t(j-1)$ for all j
    - **Move the head one cell to the left and readjust indices** -1 is the new 0, and $j - 1$ the new j
  - $\delta = (r, a, R, s)$, $q = r$, $q' = s$, $t(0) = a$, and $t'(j) = t(j+1)$ for all j.
    - **Move the head one cell to the right and readjust indices** 1 is the new 0, and $j + 1$ the new j
- $C \longrightarrow_M C'$ iff $C \xrightarrow{\delta} C'$ for some $\delta \in \Delta$

Turing computability

Definition (Turing computable functions)

A (partial) function $f : \mathbb{N} \to \mathbb{N}$ is **Turing computable** if there is a Turing machine M s.t. for any $n \in \mathbb{N}$, letting C be the (unique) initial configuration with $L(C) = n$,

$$f(n) \text{ is defined} \quad \text{iff} \quad \text{there is a final configuration } C' \text{ s.t. } L(C') = f(n) \text{ and } C \xrightarrow{*}_M C'$$

Turing computability

Definition (Turing computable functions)

A (partial) function $f : \mathbb{N} \to \mathbb{N}$ is **Turing computable** if there is a Turing machine M s.t. for any $n \in \mathbb{N}$, letting C be the (unique) initial configuration with $L(C) = n$,

$$f(n) \text{ is defined} \quad \text{iff} \quad \text{there is a final configuration } C' \text{ s.t. } L(C') = f(n) \text{ and } C \xrightarrow{*}_M C'$$

- Since the machine is deterministic and there is no move out of a final configuration, there is at most one final configuration C' s.t. $C \xrightarrow{*}_M C'$

Primitive recursive coding of configurations

- Let $M = (Q, \Delta)$ be a TM with $Q = \{q_0, \ldots, q_l\}$

Primitive recursive coding of configurations

- Let $M = (Q, \Delta)$ be a TM with $Q = \{q_0, \ldots, q_l\}$
- Configuration $C = (q_j, t)$ is encoded by the number $pair(j, pair(L(C), R(C)))$

Primitive recursive coding of configurations

- Let $M = (Q, \Delta)$ be a TM with $Q = \{q_0, \ldots, q_l\}$
- Configuration $C = (q_j, t)$ is encoded by the number $pair(j, pair(L(C), R(C)))$
- State of a configuration encoded by n is given by $state(n) = fst(n)$

Primitive recursive coding of configurations

- Let $M = (Q, \Delta)$ be a TM with $Q = \{q_0, \ldots, q_l\}$
- Configuration $C = (q_j, t)$ is encoded by the number $pair(j, pair(L(C), R(C)))$
- State of a configuration encoded by n is given by $state(n) = fst(n)$
- If C is coded by n, $L(C)$ is coded by $left(n) = fst(snd(n))$

Primitive recursive coding of configurations

- Let $M = (Q, \Delta)$ be a TM with $Q = \{q_0, \ldots, q_l\}$
- Configuration $C = (q_j, t)$ is encoded by the number $pair(j, pair(L(C), R(C)))$
- State of a configuration encoded by n is given by $state(n) = fst(n)$
- If C is coded by n, $L(C)$ is coded by $left(n) = fst(snd(n))$
- If C is coded by n, $R(C)$ is coded by $right(n) = snd(snd(n))$

Primitive recursive coding of configurations

- Let $M = (Q, \Delta)$ be a TM with $Q = \{q_0, \ldots, q_l\}$
- Configuration $C = (q_j, t)$ is encoded by the number $pair(j, pair(L(C), R(C)))$
- State of a configuration encoded by n is given by $state(n) = fst(n)$
- If C is coded by n, $L(C)$ is coded by $left(n) = fst(snd(n))$
- If C is coded by n, $R(C)$ is coded by $right(n) = snd(snd(n))$
- $config(n) := (0 \leqslant state(n) \leqslant l)$ says that n encodes a configuration of M

Primitive recursive coding of configurations

- Let $M = (Q, \Delta)$ be a TM with $Q = \{q_0, \ldots, q_l\}$
- Configuration $C = (q_j, t)$ is encoded by the number $pair(j, pair(L(C), R(C)))$
- State of a configuration encoded by n is given by $state(n) = fst(n)$
- If C is coded by n, $L(C)$ is coded by $left(n) = fst(snd(n))$
- If C is coded by n, $R(C)$ is coded by $right(n) = snd(snd(n))$
- $config(n) := (0 \leqslant state(n) \leqslant l)$ says that n encodes a configuration of M
- $initial(n) := (state(n) = 0 \wedge right(n) = 0)$ says that n encodes an initial configuration

Primitive recursive coding of configurations

- Let $M = (Q, \Delta)$ be a TM with $Q = \{q_0, \ldots, q_l\}$
- Configuration $C = (q_j, t)$ is encoded by the number $pair(j, pair(L(C), R(C)))$
- State of a configuration encoded by n is given by $state(n) = fst(n)$
- If C is coded by n, $L(C)$ is coded by $left(n) = fst(snd(n))$
- If C is coded by n, $R(C)$ is coded by $right(n) = snd(snd(n))$
- $config(n) := (0 \leqslant state(n) \leqslant l)$ says that n encodes a configuration of M
- $initial(n) := (state(n) = 0 \wedge right(n) = 0)$ says that n encodes an initial configuration
- $final(n) := (state(n) = 1 \wedge right(n) = 0)$ says that n encodes a final configuration.

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, b, q_t) \in \Delta$ with $b \in \{0, 1\}$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, b, q_t) \in \Delta$ with $b \in \{0, 1\}$
- Let c and c' encode two configurations C and C' of M

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, b, q_t) \in \Delta$ with $b \in \{0, 1\}$
- Let c and c' encode two configurations C and C' of M
- $step_\delta(c, c')$ is defined as follows:

$$config(c) \wedge config(c') \wedge state(c) = s \wedge state(c') = t$$
$$\wedge lsb(left(c)) = a \wedge$$
$$left(c') = left(c) + b - a \wedge right(c') = right(c)$$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, b, q_t) \in \Delta$ with $b \in \{0, 1\}$
- Let c and c' encode two configurations C and C' of M
- $step_\delta(c, c')$ is defined as follows:

$$config(c) \wedge config(c') \wedge state(c) = s \wedge state(c') = t$$
$$\wedge lsb(left(c)) = a \wedge$$
$$left(c') = left(c) + b - a \wedge right(c') = right(c)$$

- **Exercise:** Verify that $step_\delta(c, c')$ is true iff $C \xrightarrow{\delta} C'$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, L, q_t) \in \Delta$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, L, q_t) \in \Delta$
- Let c and c' encode two configurations C and C' of M

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, L, q_t) \in \Delta$
- Let c and c' encode two configurations C and C' of M
- $step_\delta(c, c')$ is defined as follows:

$$config(c) \wedge config(c') \wedge state(c) = s \wedge state(c') = t$$
$$\wedge lsb(left(c)) = a \wedge$$
$$2 \cdot left(c') + a = left(c) \wedge right(c') = 2 \cdot right(c) + a$$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, L, q_t) \in \Delta$
- Let c and c' encode two configurations C and C' of M
- $step_\delta(c, c')$ is defined as follows:

$$config(c) \wedge config(c') \wedge state(c) = s \wedge state(c') = t$$
$$\wedge lsb(left(c)) = a \wedge$$
$$2 \cdot left(c') + a = left(c) \wedge right(c') = 2 \cdot right(c) + a$$

- **Exercise:** Verify that $step_\delta(c, c')$ is true iff $C \xrightarrow{\delta} C'$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, R, q_t) \in \Delta$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, R, q_t) \in \Delta$
- Let c and c' encode two configurations C and C' of M

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, R, q_t) \in \Delta$
- Let c and c' encode two configurations C and C' of M
- $step_\delta(c, c')$ is defined as follows:

$$\exists b \leqslant 1 \left\{ \begin{array}{c} config(c) \wedge config(c') \wedge state(c) = s \wedge state(c') = t \\ \wedge\, lsb(left(c)) = a \wedge lsb(right(c)) = b \wedge \\ left(c') = 2 \cdot left(c) + b \wedge 2 \cdot right(c') + b = right(c) \end{array} \right\}$$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, R, q_t) \in \Delta$
- Let c and c' encode two configurations C and C' of M
- $step_\delta(c, c')$ is defined as follows:

$$\exists b \leqslant 1 \left\{ \begin{array}{c} config(c) \ \wedge \ config(c') \ \wedge \ state(c) = s \ \wedge \ state(c') = t \\ \wedge \ lsb(left(c)) = a \ \wedge \ lsb(right(c)) = b \wedge \\ left(c') = 2 \cdot left(c) + b \ \wedge \ 2 \cdot right(c') + b = right(c) \end{array} \right\}$$

- **Exercise:** Verify that $step_\delta(c, c')$ is true iff $C \xrightarrow{\delta} C'$

Primitive recursive coding of transitions

- Let $M = (Q, \Delta)$ be a TM, and $\delta = (q_s, a, R, q_t) \in \Delta$
- Let c and c' encode two configurations C and C' of M
- $step_\delta(c, c')$ is defined as follows:

$$\exists b \leqslant 1 \left\{ \begin{array}{c} config(c) \ \wedge \ config(c') \ \wedge \ state(c) = s \ \wedge \ state(c') = t \\ \wedge \ lsb(left(c)) = a \ \wedge \ lsb(right(c)) = b \wedge \\ left(c') = 2 \cdot left(c) + b \ \wedge \ 2 \cdot right(c') + b = right(c) \end{array} \right\}$$

- **Exercise:** Verify that $step_\delta(c, c')$ is true iff $C \xrightarrow{\delta} C'$
- $step_M(c, c') \Leftrightarrow \bigvee_{\delta \in \Delta} step_\delta(c, c')$.

Primitive recursive coding of computations

- Let $M = (Q, \Delta)$ be a TM

Primitive recursive coding of computations

- Let $M = (Q, \Delta)$ be a TM
- A **terminating run** of M on input n is given by a sequence of configurations $C_1, \ldots, C_k$ and a number m such that:

Primitive recursive coding of computations

- Let $M = (Q, \Delta)$ be a TM
- A **terminating run** of M on input n is given by a sequence of configurations $C_1, \ldots, C_k$ and a number m such that:
  - C_1 is an initial configuration with $L(C_1) = n$;

Primitive recursive coding of computations

- Let $M = (Q, \Delta)$ be a TM
- A **terminating run** of M on input n is given by a sequence of configurations $C_1, \ldots, C_k$ and a number m such that:
  - C_1 is an initial configuration with $L(C_1) = n$;
  - C_k is a final configuration, with $L(C_k) = m$; and

Primitive recursive coding of computations

- Let $M = (Q, \Delta)$ be a TM
- A **terminating run** of M on input n is given by a sequence of configurations $C_1, \ldots, C_k$ and a number m such that:
  - C_1 is an initial configuration with $L(C_1) = n$;
  - C_k is a final configuration, with $L(C_k) = m$; and
  - for all $i < k$, $C_i \longrightarrow_M C_{i+1}$

Primitive recursive coding of computations

- Let $M = (Q, \Delta)$ be a TM
- A **terminating run** of M on input n is given by a sequence of configurations $C_1, \ldots, C_k$ and a number m such that:
  - C_1 is an initial configuration with $L(C_1) = n$;
  - C_k is a final configuration, with $L(C_k) = m$; and
  - for all $i < k$, $C_i \longrightarrow_M C_{i+1}$
- The primitive recursive predicate $run_M(n, r)$ says that r codes a terminating run of M on input n:

$$\exists s, k, m \leqslant r \left\{ \begin{array}{c} r = pair(m, s) \ \wedge \ Seq(s) \wedge k = ln(s) \\ \wedge initial((s)_1) \ \wedge \ final((s)_k) \\ \wedge left((s)_1) = n \ \wedge \ left((s)_k) = m \\ \wedge (\forall i < k)[step_M((s)_i, (s)_{i+1})] \end{array} \right\}$$

Primitive recursive coding of computations

- Let $M = (Q, \Delta)$ be a TM
- A **terminating run** of M on input n is given by a sequence of configurations $C_1, \ldots, C_k$ and a number m such that:
  - C_1 is an initial configuration with $L(C_1) = n$;
  - C_k is a final configuration, with $L(C_k) = m$; and
  - for all $i < k$, $C_i \longrightarrow_M C_{i+1}$
- The primitive recursive predicate $run_M(n, r)$ says that r codes a terminating run of M on input n:

$$\exists s, k, m \leqslant r \left\{ \begin{array}{c} r = pair(m, s) \ \wedge \ Seq(s) \wedge k = ln(s) \\ \wedge initial((s)_1) \ \wedge \ final((s)_k) \\ \wedge left((s)_1) = n \ \wedge \ left((s)_k) = m \\ \wedge (\forall i < k)[step_M((s)_i, (s)_{i+1})] \end{array} \right\}$$

- If r encodes a run of M, $fst(r)$ returns the output of the run.

Turing computable functions are partial recursive

Theorem

Any Turing computable (partial) function $f : \mathbb{N} \to \mathbb{N}$ is also (partial) recursive.

Turing computable functions are partial recursive

Theorem

Any Turing computable (partial) function $f : \mathbb{N} \to \mathbb{N}$ is also (partial) recursive.

Proof.

Suppose f is computed by a Turing machine M. Then for any n for which f is defined, there is a least number r that encodes a terminating computation of M on input n, and its output is $f(n)$. Thus we can define f as follows:

$$f(n) = fst\left[\mu r.run_M(n, r)\right].$$

If f is not defined on n, the RHS in the above is not defined either, so the equation remains true. Since run_M and fst are primitive recursive, it follows that f is partial recursive. ⊣

A normal form for partial recursive functions

Theorem (Kleene's normal form theorem)

A function f is (partial) recursive iff there is a primitive recursive predicate T s.t. $f(n) = fst(\mu r.T(n, r))$ for all n.

A normal form for partial recursive functions

Theorem (Kleene's normal form theorem)

A function f is (partial) recursive iff there is a primitive recursive predicate T s.t. $f(n) = fst(\mu r.T(n, r))$ for all n.

Proof.

Any function of the form $fst(\mu r.T(n, r))$ with primitive recursive T is recursive. In the other direction, given a recursive function f, translate it to the corresponding Turing machine M, and express $f(n)$ as $fst\left[\mu r.run_M(n, r)\right]$. ⊣