



Computability Theory

Aart Middeldorp

Outline

- 1. Summary of Previous Lecture**
- 2. Gödel Numbering**
- 3. Course-of-Values Recursion**
- 4. Ackermann Function**
- 5. Diagonalization**
- 6. Total Recursive Functions**
- 7. Summary**

Definition

class **PR** of **primitive recursive functions** is smallest class of total functions $f: \mathbb{N}^n \rightarrow \mathbb{N}$ that contains all **initial functions**

- ▶ **zero** $z(x) = 0$
- ▶ **successor** $s(x) = x + 1$
- ▶ **projection** $\pi_i^n(x_1, \dots, x_n) = x_i$ for all $n \geq 1$ and $1 \leq i \leq n$

and is closed under **composition**

- ▶ $f(\vec{x}) = g(h_1(\vec{x}), \dots, h_m(\vec{x})) \in \text{PR}$ for all $g: \mathbb{N}^m \rightarrow \mathbb{N} \in \text{PR}$ and $h_1, \dots, h_m: \mathbb{N}^n \rightarrow \mathbb{N} \in \text{PR}$

and **primitive recursion**

- ▶ $f(x, \vec{y}): \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ defined by

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

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

belongs to PR for all $g: \mathbb{N}^n \rightarrow \mathbb{N} \in \text{PR}$ and $h: \mathbb{N}^{n+2} \rightarrow \mathbb{N} \in \text{PR}$

Definition

predicate $P: \mathbb{N}^n \rightarrow \mathbb{B}$ is primitive recursive if its **characteristic function** $\chi_P: \mathbb{N}^n \rightarrow \mathbb{N}$

$$\chi_P(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } P(x_1, \dots, x_n) \\ 0 & \text{otherwise} \end{cases}$$

is primitive recursive

Lemma

PR is closed under **iteration**, **case analysis** and **bounded minimization**

Definitions

► **pairing** function $\pi(x, y) = 2^x(2y + 1) \dot{-} 1$

► **extraction** functions

$$\pi_1(z) = (\mu x \leq z) (\exists y \leq z) [z = \pi(x, y)] \quad \pi_2(z) = (\mu y \leq z) (\exists x \leq z) [z = \pi(x, y)]$$

Part I: Recursive Function Theory

Ackermann function, bounded minimization, bounded recursion, course-of-values recursion, diagonalization, diophantine sets, elementary functions, fixed point theorem, Fibonacci numbers, Gödel numbering, Gödel's β function, Grzegorczyk hierarchy, loop programs, minimization, normal form theorem, partial recursive functions, primitive recursion, recursive enumerability, recursive inseparability, s-m-n theorem, total recursive functions, undecidability, while programs

Part II: Combinatory Logic

abstraction, arithmetization, CL-representability, combinators, combinatorial completeness, Church numerals, Church-Rosser theorem, Curry-Howard isomorphism, fixed point theorem, intuitionistic propositional logic, normalization theorem, termination, typing, undecidability, Z property

Outline

1. Summary of Previous Lecture

2. Gödel Numbering

3. Course-of-Values Recursion

4. Ackermann Function

5. Diagonalization

6. Total Recursive Functions

7. Summary

Definitions

- ▶ **Gödel number** of sequence $x_1, \dots, x_n$: $\langle x_1, \dots, x_n \rangle = p_0^n \times p_1^{x_1} \times \dots \times p_n^{x_n}$
- ▶ $(x)_i = (\mu j < x) \neg (p_i^{j+1} \mid x)$
- ▶ $\text{len}(x) = (x)_0$
- ▶ $\text{seq}(x) \iff x > 0 \wedge (\forall i \leq x) [(x)_i \neq 0 \implies i \leq \text{len}(x)]$
- ▶ $x ; y = p_0^{\text{len}(x) + \text{len}(y)} \times \prod_{i < \text{len}(x)} p_{i+1}^{(x)_{i+1}} \times \prod_{i < \text{len}(y)} p_{\text{len}(x) + i + 1}^{(y)_{i+1}}$

Lemma

- ▶ $x = \langle x_1, \dots, x_n \rangle \implies \text{len}(x) = n \wedge (x)_i = x_i \text{ for all } 1 \leq i \leq n$
- ▶ $\text{seq}(x) \wedge \text{len}(x) = n \implies x = \langle (x)_1, \dots, (x)_n \rangle$
- ▶ $x = \langle x_1, \dots, x_n \rangle \wedge y = \langle y_1, \dots, y_m \rangle \implies x ; y = \langle x_1, \dots, x_n, y_1, \dots, y_m \rangle$

Outline

1. Summary of Previous Lecture
2. Gödel Numbering
- 3. Course-of-Values Recursion**
4. Ackermann Function
5. Diagonalization
6. Total Recursive Functions
7. Summary

Definition

if $f: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ then $\tilde{f}(x, \vec{y}) = \langle f(0, \vec{y}), \dots, f(x, \vec{y}) \rangle$

Lemma

if $g: \mathbb{N}^n \rightarrow \mathbb{N}$ and $h: \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ are primitive recursive then so is $f: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ defined as

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

$$f(x+1, \vec{y}) = h(\tilde{f}(x, \vec{y}), x, \vec{y})$$

Proof

$f(x, \vec{y}) = (\tilde{f}(x, \vec{y}))_{x+1}$ and $\tilde{f}$ is primitive recursive:

$$\tilde{f}(0, \vec{y}) = \langle f(0, \vec{y}) \rangle = \langle g(\vec{y}) \rangle$$

$$\tilde{f}(x+1, \vec{y}) = \tilde{f}(x, \vec{y}) ; \langle f(x+1, \vec{y}) \rangle = \tilde{f}(x, \vec{y}) ; \langle h(\tilde{f}(x, \vec{y}), x, \vec{y}) \rangle = h'(\tilde{f}(x, \vec{y}), x, \vec{y})$$

with $h'(x, y, \vec{z}) = x ; \langle h(x, y, \vec{z}) \rangle$

Outline

1. Summary of Previous Lecture
2. Gödel Numbering
3. Course-of-Values Recursion
- 4. Ackermann Function**
5. Diagonalization
6. Total Recursive Functions
7. Summary

Definition

$$\text{ack}(0, y) = y + 1$$

$$\text{ack}(x + 1, 0) = \text{ack}(x, 1)$$

$$\text{ack}(x + 1, y + 1) = \text{ack}(x, \text{ack}(x + 1, y))$$



Example

- ▶ $\text{ack}(1, 3) = 5$
- ▶ $\text{ack}(2, 2) = 7$
- ▶ $\text{ack}(3, 4) = 125$
- ▶ $\text{ack}(4, 1) = 65533$

Lemma

Ackermann function is well-defined

Example (cont'd)

► $\text{ack}(4, 3) = 2^{2^{65533}} - 3$

Remarks

- Ackermann function grows very fast
- Ackermann function is used as compiler benchmark
- inverse of Ackermann function appears in complexity analysis of certain algorithms

Lemma

- 1 $\text{ack}(x, y) > y$
- 2 $\text{ack}(x, y + 1) > \text{ack}(x, y)$
- 3 $\text{ack}(x + 1, y) \geq \text{ack}(x, y + 1)$
- 4 $\text{ack}(x + 2, y) > \text{ack}(x, 2y)$

► skip proofs

Lemma 1

► $\text{ack}(x, y) > y$

Proof

induction on x

► $\text{ack}(0, y) = y + 1 > y$

► induction on y

$$\text{ack}(x + 1, 0) = \text{ack}(x, 1) > 1 > 0$$

$$\text{ack}(x + 1, y + 1) = \text{ack}(x, \text{ack}(x + 1, y)) > \text{ack}(x + 1, y) \geq y + 1$$

Lemma 2

$$\triangleright \text{ack}(x, y + 1) > \text{ack}(x, y)$$

Proof

induction on x

$$\text{ack}(0, y + 1) = y + 2 > y + 1 = \text{ack}(0, y)$$

$$\text{ack}(x + 1, y + 1) = \text{ack}(x, \text{ack}(x + 1, y)) > \text{ack}(x + 1, y)$$

Lemma 3

$$\triangleright \text{ack}(x+1, y) \geq \text{ack}(x, y+1) > \text{ack}(x, y)$$

Proof

induction on x

$$\triangleright \text{ack}(1, y) \geq y+2 = \text{ack}(0, y+1) \text{ by induction on } y$$

$$\text{ack}(1, 0) = \text{ack}(0, 1) = 2$$

$$\text{ack}(1, y+1) = \text{ack}(0, \text{ack}(1, y)) \geq \text{ack}(0, y+2)$$

$\triangleright$ induction on y

$$\text{ack}(x+2, 0) = \text{ack}(x+1, 1)$$

$$\begin{aligned} \text{ack}(x+2, y+1) &= \text{ack}(x+1, \text{ack}(x+2, y)) \geq \text{ack}(x+1, \text{ack}(x+1, y+1)) \\ &\geq \text{ack}(x+1, \text{ack}(x, y+2)) > \text{ack}(x+1, y+2) \end{aligned}$$

Lemma 4

$$\triangleright \text{ack}(x + 2, y) > \text{ack}(x, 2y)$$

Proof

induction on x

$$\triangleright \text{ack}(2, y) > \text{ack}(0, 2y) \text{ by induction on } y$$

$$\text{ack}(2, 0) \geq \text{ack}(1, 1) \geq \text{ack}(0, 2) > \text{ack}(0, 1) > \text{ack}(0, 0)$$

$$\text{ack}(2, y + 1) = \text{ack}(1, \text{ack}(2, y)) > \text{ack}(1, 2y + 1) \geq \text{ack}(0, 2y + 2)$$

$$\triangleright \text{induction on } y$$

$$\text{ack}(x + 3, 0) \geq \text{ack}(x + 2, 1) \geq \text{ack}(x + 1, 2) > \text{ack}(x + 1, 0)$$

$$\text{ack}(x + 3, y + 1) = \text{ack}(x + 2, \text{ack}(x + 3, y)) > \text{ack}(x + 2, \text{ack}(x + 1, 2y))$$

$$\geq \text{ack}(x + 2, \text{ack}(x, 2y + 1)) > \text{ack}(x + 2, 2y + 1)$$

$$\geq \text{ack}(x + 1, 2y + 2)$$

Lemma

$\forall$ primitive recursive function $f: \mathbb{N}^n \rightarrow \mathbb{N}$ $\exists$ constant $c \in \mathbb{N}$ such that

$$f(x_1, \dots, x_n) < \text{ack}(c, \max \{x_1, \dots, x_n\})$$

Proof

induction on definition of primitive recursive functions

► initial functions

$$z(x) = 0 < x + 1 = \text{ack}(0, x)$$

$$s(x) = x + 1 = \text{ack}(0, x) < \text{ack}(1, x)$$

$$\pi_i^n(x_1, \dots, x_n) = x_i \leq \max \{x_1, \dots, x_n\} < \text{ack}(0, \max \{x_1, \dots, x_n\})$$

Lemma

$\forall$ primitive recursive function $f: \mathbb{N}^n \rightarrow \mathbb{N}$ $\exists$ constant $c \in \mathbb{N}$ such that

$$f(x_1, \dots, x_n) < \text{ack}(c, \max \{x_1, \dots, x_n\})$$

Proof (cont'd)

induction on definition of primitive recursive functions

► **composition** $f(\vec{x}) = g(h_1(\vec{x}), \dots, h_m(\vec{x}))$

$$g(\vec{y}) < \text{ack}(c_0, \max \{\vec{y}\}) \quad \text{and} \quad h_i(\vec{x}) < \text{ack}(c_i, \max \{\vec{x}\}) \quad \text{for all } 1 \leq i \leq m$$

$$c' = \max \{c_0 + 1, c_1, \dots, c_m\}$$

$$\begin{aligned} f(\vec{x}) &< \text{ack}(c_0, \max \{h_1(\vec{x}), \dots, h_m(\vec{x})\}) \\ &< \text{ack}(c_0, \max \{\text{ack}(c_1, \max \{\vec{x}\}), \dots, \text{ack}(c_m, \max \{\vec{x}\})\}) \\ &= \text{ack}(c_0, \text{ack}(\max \{c_1, \dots, c_m\}, \max \{\vec{x}\})) \\ &\leq \text{ack}(c' - 1, \text{ack}(c', \max \{\vec{x}\})) = \text{ack}(c', \max \{\vec{x}\} + 1) \leq \text{ack}(c' + 1, \max \{\vec{x}\}) \end{aligned}$$

Lemma

$\forall$ primitive recursive function $f: \mathbb{N}^n \rightarrow \mathbb{N}$ $\exists$ constant $c \in \mathbb{N}$ such that

$$f(x_1, \dots, x_n) < \text{ack}(c, \max \{x_1, \dots, x_n\})$$

Proof (cont'd)

induction on definition of primitive recursive functions

► **primitive recursion** $f(0, \vec{y}) = g(\vec{y})$

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

$$g(\vec{y}) < \text{ack}(c_1, \max \{\vec{y}\}) \quad \text{and} \quad h(a, b, \vec{y}) < \text{ack}(c_2, \max \{a, b, \vec{y}\})$$

$$c' = \max \{c_1, c_2 + 1\}$$

$$f(x, \vec{y}) < \text{ack}(c', x + \max \{\vec{y}\}) \text{ by induction on } x$$

$$f(x, \vec{y}) < \text{ack}(c', 2 \cdot \max \{x, \vec{y}\}) < \text{ack}(c' + 2, \max \{x, \vec{y}\})$$

Theorem

Ackermann function is **not** primitive recursive

Proof

if Ackermann function is primitive recursive then

$$\text{ack}(x, y) < \text{ack}(c, \max\{x, y\})$$

for some constant c and thus

$$\text{ack}(c, c) < \text{ack}(c, \max\{c, c\}) = \text{ack}(c, c)$$



Outline

1. Summary of Previous Lecture
2. Gödel Numbering
3. Course-of-Values Recursion
4. Ackermann Function
- 5. Diagonalization**
6. Total Recursive Functions
7. Summary

Definition

index $\ulcorner f \urcorner \in \mathbb{N}$ of **derivation** of primitive recursive function f is defined inductively:

- ▶ $\ulcorner z \urcorner = \langle 0 \rangle$
- ▶ $\ulcorner s \urcorner = \langle 1 \rangle$
- ▶ $\ulcorner \pi_i^n \urcorner = \langle 2, n, i \rangle$
- ▶ $\ulcorner f \urcorner = \langle 3, \ulcorner g \urcorner, \ulcorner h_1 \urcorner, \dots, \ulcorner h_m \urcorner \rangle$ if f is obtained by composing g and $h_1, \dots, h_m$
- ▶ $\ulcorner f \urcorner = \langle 4, \ulcorner g \urcorner, \ulcorner h \urcorner \rangle$ if f is obtained by primitive recursion from g and h

Examples

- ▶ $\ulcorner + \urcorner = \langle 4, \langle 2, 1, 1 \rangle, \langle 3, \langle 1 \rangle, \langle 2, 3, 1 \rangle \rangle \rangle$
- ▶ $397065375000 = 2^3 \times 3^3 \times 5^6 \times 7^6 = \langle 3, \langle 1 \rangle, \langle 1 \rangle \rangle = \ulcorner s \circ s \urcorner$

Remark (key insight)

from index we can reconstruct function

(details in later lecture)

Corollary

$\exists$ computable list $f_0, f_1, f_2, f_3, \dots$ of unary primitive recursive functions

Diagonalization

$f_0(0)$	$f_0(1)$	$f_0(2)$	$f_0(3)$	$\dots$
$f_1(0)$	$f_1(1)$	$f_1(2)$	$f_1(3)$	$\dots$
$f_2(0)$	$f_2(1)$	$f_2(2)$	$f_2(3)$	$\dots$
$f_3(0)$	$f_3(1)$	$f_3(2)$	$f_3(3)$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$$g(x) = f_x(x) + 1$$

Corollary

function g is computable but not primitive recursive

Outline

1. Summary of Previous Lecture
2. Gödel Numbering
3. Course-of-Values Recursion
4. Ackermann Function
5. Diagonalization
- 6. Total Recursive Functions**
7. Summary

Definition

class **R** of **recursive functions** is smallest class of **total** functions that contains all initial functions and is closed under composition, primitive recursion, and **(unbounded) minimization**:

$$(\mu i) P(i, \vec{y}) = \min \{i \mid P(i, \vec{y})\} \in R$$

for all $P: \mathbb{N}^{n+1} \rightarrow \mathbb{B}$ such that $\chi_P \in R$ and $\forall \vec{y} \exists x P(x, \vec{y})$

Theorem

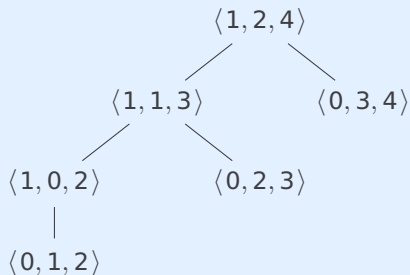
Ackermann function is recursive

$$\text{ack}(0, y) = y + 1$$

$$\text{ack}(x + 1, 0) = \text{ack}(x, 1)$$

$$\text{ack}(x + 1, y + 1) = \text{ack}(x, \text{ack}(x + 1, y))$$

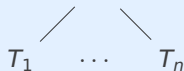
$\text{ack}(1, 2) = 4$:



$\langle x, y, z \rangle$ denotes $\text{ack}(x, y) = z$

Definition

encode computation tree $T = \langle x, y, z \rangle$ as number $\ulcorner T \urcorner = \langle \langle x, y, z \rangle, \ulcorner T_1 \urcorner, \dots, \ulcorner T_n \urcorner \rangle$



Lemma

$\exists$ primitive recursive predicate $P: \mathbb{N} \rightarrow \mathbb{B}$ such that

$$P(x) \iff x \text{ encodes correct computation tree}$$

Corollary

- ▶ $\text{ack}(x, y) = z \iff \exists t \text{ such that } P(t) \text{ and } (t)_1 = \langle x, y, z \rangle$
- ▶ $\text{ack}(x, y) = (\mu t) (P(t) \wedge (t)_{1,1} = x \wedge (t)_{1,2} = y) 1, 3$

Proof

- ▶ $P(x) \iff \text{seq}(x) \wedge \text{seq}((x)_1) \wedge \text{len}((x)_1) = 3 \wedge (A(x) \vee B(x) \vee C(x))$
- ▶ $A(x) \iff \text{len}(x) = 1 \wedge (x)_{1,1} = 0 \wedge (x)_{1,3} = s((x)_{1,2})$
- ▶ $B(x) \iff \text{len}(x) = 2 \wedge (x)_{1,1} > 0 \wedge (x)_{1,2} = 0 \wedge (x)_{2,1,1} = p((x)_{1,1}) \wedge (x)_{2,1,2} = 1 \wedge (x)_{2,1,3} = (x)_{1,3} \wedge P((x)_2)$
- ▶ $C(x) \iff \text{len}(x) = 3 \wedge (x)_{1,1} > 0 \wedge (x)_{1,2} > 0 \wedge (x)_{2,1,1} = (x)_{1,1} \wedge (x)_{2,1,2} = p((x)_{1,2}) \wedge (x)_{3,1,1} = p((x)_{1,1}) \wedge (x)_{3,1,2} = (x)_{2,1,3} \wedge (x)_{3,1,3} = (x)_{1,3} \wedge P((x)_2) \wedge P((x)_3)$

$$T = \langle x, y, z \rangle \quad \ulcorner T \urcorner = \langle \langle x, y, z \rangle, \ulcorner T_1 \urcorner, \dots, \ulcorner T_n \urcorner \rangle$$

```
graph TD; T["T = <x, y, z>"] --- T1["T_1"]; T --- Ellipsis["..."]; T --- Tn["T_n"]
```

Outline

1. Summary of Previous Lecture
2. Gödel Numbering
3. Course-of-Values Recursion
4. Ackermann Function
5. Diagonalization
6. Total Recursive Functions
- 7. Summary**

Important Concepts

- ▶ Ackermann function
- ▶ course-of-values recursion
- ▶ diagonalization
- ▶ Gödel numbering
- ▶ index
- ▶ R
- ▶ recursive function
- ▶ (unbounded) minimization

homework for October 19