Recursion Theory

Joost J. Joosten

Institute for Logic Language and Computation

University of Amsterdam

Plantage Muidergracht 24

1018 TV Amsterdam

Room P 3.26, +31 20 5256095

jjoosten@phil.uu.nl

www.phil.uu.nl/~jjoosten

The fixed point theorem

- Kleene's Fixed point theorem (1938) is very important!

The fixed point theorem

- Kleene's Fixed point theorem (1938) is very important!
- For all computable f, there exists a k such that $\varphi_{f(k)} = \varphi_k$

The fixed point theorem

- Kleene's Fixed point theorem (1938) is very important!
- For all computable f, there exists a k such that $\varphi_{f(k)} = \varphi_k$
- Proof: consider the function h such that $\varphi_{h(x)} = \varphi_{\varphi_x(x)}$

The fixed point theorem

- Kleene's Fixed point theorem (1938) is very important!
- For all computable f, there exists a k such that $\varphi_{f(k)} = \varphi_k$
- Proof: consider the function h such that $\varphi_{h(x)} = \varphi_{\varphi_x(x)}$ (again this diagonal!)

The fixed point theorem

- Kleene's Fixed point theorem (1938) is very important!
- For all computable f, there exists a k such that $\varphi_{f(k)} = \varphi_k$
- Proof: consider the function h such that $\varphi_{h(x)} = \varphi_{\varphi_x(x)}$ (again this diagonal!)
- Note: we do not say $h(x) = \varphi_x(x)$

The fixed point theorem

- Kleene's Fixed point theorem (1938) is very important!
- For all computable f, there exists a k such that $\varphi_{f(k)} = \varphi_k$
- Proof: consider the function h such that $\varphi_{h(x)} = \varphi_{\varphi_x(x)}$ (again this diagonal!)
- Note: we do not say $h(x) = \varphi_x(x)$
- $f \circ h$ has code e, that is, $(f \circ h)(x) = \varphi_e(x)$

The fixed point theorem

- Kleene's Fixed point theorem (1938) is very important!
- For all computable f, there exists a k such that $\varphi_{f(k)} = \varphi_k$
- Proof: consider the function h such that $\varphi_{h(x)} = \varphi_{\varphi_x(x)}$ (again this diagonal!)
- Note: we do not say $h(x) = \varphi_x(x)$
- $f \circ h$ has code e, that is, $(f \circ h)(x) = \varphi_e(x)$
- So, we can take k to be $h(e)$

The fixed point theorem

- Kleene's Fixed point theorem (1938) is very important!
- For all computable f, there exists a k such that $\varphi_{f(k)} = \varphi_k$
- Proof: consider the function h such that $\varphi_{h(x)} = \varphi_{\varphi_x(x)}$ (again this diagonal!)
- Note: we do not say $h(x) = \varphi_x(x)$
- $f \circ h$ has code e, that is, $(f \circ h)(x) = \varphi_e(x)$
- So, we can take k to be $h(e)$ (here we use that h should be total!)

Fixed points

- Note: the diagonal construction $\varphi_x(x)$ allows us to view the very same number both as the input of a program and as a program

Fixed points

- Note: the diagonal construction $\varphi_x(x)$ allows us to view the very same number both as the input of a program and as a program
- This allows some sort of self reference!

Fixed points

- Note: the diagonal construction $\varphi_x(x)$ allows us to view the very same number both as the input of a program and as a program
- This allows some sort of self reference!
- Fixed points and a map of Amsterdam

Computable approximations

- We shall see (more or less, we have already seen) that $\varphi_e(x)\downarrow$ and $\varphi_e(x) = y$ is incomputable

Computable approximations

- We shall see (more or less, we have already seen) that $\varphi_e(x) \downarrow$ and $\varphi_e(x) = y$ is incomputable
- However, $\varphi_{e,s}(x) = y$ is of course computable

Computable approximations

- We shall see (more or less, we have already seen) that $\varphi_e(x)\downarrow$ and $\varphi_e(x) = y$ is incomputable
- However, $\varphi_{e,s}(x) = y$ is of course computable
- We shall use these approximations later

Recap of previous lecture

- Coding: representing programs by numbers

Recap of previous lecture

- Coding: representing programs by numbers
- Enumeration Theorem/Universal Turing Machine

Recap of previous lecture

- Coding: representing programs by numbers
- Enumeration Theorem/Universal Turing Machine
- Padding Lemma
- S_n^m-theorem

Recap of previous lecture

- Coding: representing programs by numbers
- Enumeration Theorem/Universal Turing Machine
- Padding Lemma
- S_n^m-theorem
- Fixed point theorem

Recap of previous lecture

- Coding: representing programs by numbers
- Enumeration Theorem/Universal Turing Machine
- Padding Lemma
- S_n^m-theorem
- Fixed point theorem
- Computable approximations of uncomputable problems

Computably enumerable sets

- We are in need of the notion of a set that is not necessarily computable, but rather computably enumerable

Computably enumerable sets

- We are in need of the notion of a set that is not necessarily computable, but rather computably enumerable
- Motivation

Computably enumerable sets

- We are in need of the notion of a set that is not necessarily computable, but rather computably enumerable
- Motivation
- Consider $\{x \in \mathbb{N} \mid$ there is a sequence of at least x 7's in the decimal expansion of π $\}$

Computably enumerable sets

- We are in need of the notion of a set that is not necessarily computable, but rather computably enumerable
- Motivation
- Consider $\{x \in \mathbb{N} \mid$ there is a sequence of at least x 7's in the decimal expansion of π $\}$
- Next, consider $\{x \in \mathbb{N} \mid$ there is a sequence of exactly x (and no more) 7's in the decimal expansion of π $\}$

Computably enumerable sets

- We are in need of the notion of a set that is not necessarily computable, but rather computably enumerable
- Motivation
- Consider $\{x \in \mathbb{N} \mid$ there is a sequence of at least x 7's in the decimal expansion of π $\}$
- Next, consider $\{x \in \mathbb{N} \mid$ there is a sequence of exactly x (and no more) 7's in the decimal expansion of π $\}$
- Although the latter is not computable, it is enumerable in an effective way.

Computably enumerable

- Formal definition of A being c.e.: it is the range of a computable function (or empty)

Computably enumerable

- Formal definition of A being c.e.: it is the range of a computable function (or empty)
- Under the CT-thesis we also are allowed to call this RE

Computably enumerable

- Formal definition of A being c.e.: it is the range of a computable function (or empty)
- Under the CT-thesis we also are allowed to call this RE
- Introduced by Emil Post (1897-1954)

Computably enumerable

- Formal definition of A being c.e.: it is the range of a computable function (or empty)
- Under the CT-thesis we also are allowed to call this RE
- Introduced by Emil Post (1897-1954) (almost proved Gödel 1, was a high-school teacher for some period of time)

Computably enumerable

- Formal definition of A being c.e.: it is the range of a computable function (or empty)
- Under the CT-thesis we also are allowed to call this RE
- Introduced by Emil Post (1897-1954) (almost proved Gödel 1, was a high-school teacher for some period of time)
- Theorem: If A is computable, it is also computably enumerable

Computably enumerable

- Formal definition of A being c.e.: it is the range of a computable function (or empty)
- Under the CT-thesis we also are allowed to call this RE
- Introduced by Emil Post (1897-1954) (almost proved Gödel 1, was a high-school teacher for some period of time)
- Theorem: If A is computable, it is also computably enumerable
- Theorem: A is computable iff both A and $\overline{A}$ are CE

Computably enumerable

- Formal definition of A being c.e.: it is the range of a computable function (or empty)
- Under the CT-thesis we also are allowed to call this RE
- Introduced by Emil Post (1897-1954) (almost proved Gödel 1, was a high-school teacher for some period of time)
- Theorem: If A is computable, it is also computably enumerable
- Theorem: A is computable iff both A and $\overline{A}$ are CE
- This is the famous complemantiation theorem.

Closure properties of CE functions

- If A and B are both CE, then also $A \cup B$ is CE

Closure properties of CE functions

- If A and B are both CE, then also $A \cup B$ is CE
- If A and B are both CE, then also $A \cap B$ is CE

Closure properties of CE functions

- If A and B are both CE, then also $A \cup B$ is CE
- If A and B are both CE, then also $A \cap B$ is CE
- We shall prove the latter formally in the workgroup

Closure properties of CE functions

- If A and B are both CE, then also $A \cup B$ is CE
- If A and B are both CE, then also $A \cap B$ is CE
- We shall prove the latter formally in the workgroup
- What about complements?

Closure properties of CE functions

- If A and B are both CE, then also $A \cup B$ is CE
- If A and B are both CE, then also $A \cap B$ is CE
- We shall prove the latter formally in the workgroup
- What about complements? (Stay tuned!)

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.
- Proof:

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.
- Proof: "$\Leftarrow$" :

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.
- Proof: "$\Leftarrow$" : easy

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.
- Proof: "$\Leftarrow$" : easy
- "$\Rightarrow$" :

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.
- Proof: "$\Leftarrow$" : easy
- "$\Rightarrow$" : if CE, then it is the range of some computable f

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.
- Proof: "$\Leftarrow$" : easy
- "$\Rightarrow$" : if CE, then it is the range of some computable f
- We use f to define a 1-1 h:

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.
- Proof: "$\Leftarrow$" : easy
- "$\Rightarrow$" : if CE, then it is the range of some computable f
- We use f to define a 1-1 h:
  - $h(0) = f(0)$

CE sets and increasing functions

- Increasing function: $f(x+1) > f(x)$
- By induction: $f(x) \geq x$
- Theorem: An infinite A is computable iff A is enumerated by an increasing computable function
- Theorem: (Exercise 5.1.11) An infinite set is CE iff it is enumerated by a one-one computable function.
- Proof: "$\Leftarrow$" : easy
- "$\Rightarrow$" : if CE, then it is the range of some computable f
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\, [\forall\, z{\leq}x\, f(y) \neq h(z)])$

How formal should we be?

- We consider again:

How formal should we be?

- We consider again:
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\, [\forall\, z{\leq}x\, f(y) \neq h(z)])$

How formal should we be?

- We consider again:
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\ [\forall\, z {\leq} x\, f(y) \neq h(z)])$
- Is this a definition of a computable function?

How formal should we be?

- We consider again:
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\, [\forall\, z \leq x\, f(y) \neq h(z)])$
- Is this a definition of a computable function?
- We shall just need to mention: by course-of-values recursion

How formal should we be?

- We consider again:
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\,[\forall\, z{\le}x\, f(y) \neq h(z)])$
- Is this a definition of a computable function?
- We shall just need to mention: by course-of-values recursion
- Is h total?

How formal should we be?

- We consider again:
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\, [\forall\, z{\leq}x\, f(y) \neq h(z)])$
- Is this a definition of a computable function?
- We shall just need to mention: by course-of-values recursion
- Is h total?
- This we need to prove in exercises!!!!!

How formal should we be?

- We consider again:
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\, [\forall\, z {\leq} x f(y) \neq h(z)])$
- Is this a definition of a computable function?
- We shall just need to mention: by course-of-values recursion
- Is h total?
- This we need to prove in exercises!!!!!
- This is the level of formality we are after

How formal should we be?

- We consider again:
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\ [\forall\, z{\leq}x\, f(y) \neq h(z)])$
- Is this a definition of a computable function?
- We shall just need to mention: by course-of-values recursion
- Is h total?
- This we need to prove in exercises!!!!!
- This is the level of formality we are after
- In case of doubt: choose the formal solution

How formal should we be?

- We consider again:
- We use f to define a 1-1 h:
  - $h(0) = f(0)$
  - $h(x+1) = f(\mu y\,[\forall\, z{\leq}x\, f(y) \neq h(z)])$
- Is this a definition of a computable function?
- We shall just need to mention: by course-of-values recursion
- Is h total?
- This we need to prove in exercises!!!!!
- This is the level of formality we are after
- In case of doubt: choose the formal solution (how much risk do you allow?)

Characterizing c.e. sets

- We introduce some notation to characterize the c.e. sets

Characterizing c.e. sets

- We introduce some notation to characterize the c.e. sets
- First: W_e, the halting set of e

Characterizing c.e. sets

- We introduce some notation to characterize the c.e. sets
- First: W_e, the halting set of e
- Next: a relation being Σ^0_1, Π^0_1 or Δ^0_1

Characterizing c.e. sets

- We introduce some notation to characterize the c.e. sets
- First: W_e, the halting set of e
- Next: a relation being Σ_1^0, Π_1^0 or Δ_1^0
- Example: for a given e, the set $\{x \mid \varphi_e(x) \downarrow\}$ is Σ_1

Characterizing c.e. sets

- We introduce some notation to characterize the c.e. sets
- First: W_e, the halting set of e
- Next: a relation being Σ^0_1, Π^0_1 or Δ^0_1
- Example: for a given e, the set $\{x \mid \varphi_e(x) \downarrow\}$ is Σ_1
- Proof: $\varphi_e(x) \downarrow$ iff $(\exists s)\, \exists y \;\; \varphi_{e,s}(x) = y$

Normal Form Theorem

- The NFT states the equivalence of the following three statements for any set A

Normal Form Theorem

- The NFT states the equivalence of the following three statements for any set A
- (1) A is c.e.

Normal Form Theorem

- The NFT states the equivalence of the following three statements for any set A
- (1) A is c.e.
- (2) A is Σ_1^0

Normal Form Theorem

- The NFT states the equivalence of the following three statements for any set A
- (1) A is c.e.
- (2) A is Σ_1^0
- (3) A is W_e for some e

Normal Form Theorem

- The NFT states the equivalence of the following three statements for any set A
- (1) A is c.e.
- (2) A is Σ^0_1
- (3) A is W_e for some e
- Proof:

Normal Form Theorem

- The NFT states the equivalence of the following three statements for any set A
- (1) A is c.e.
- (2) A is Σ_1^0
- (3) A is W_e for some e
- Proof:
- (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1)