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`

Enrollment

=============================

- de studenten zonder coll.krt.nr kunnen alleen ingeschreven worden als ze daadwerkelijk ingeschreven staan;
- Scorelle: onduidelijk welke opleiding
- Dorrestijn: idem
- Tom, wsch Kemper: idem
- Bashan Michel: idem
- Nina: idem
- Neutel: idem
- Tsai: idem

Extra announcements

- Next friday, lecture by Sebastiaan Terwijn on the Medvedev Lattice
  See
  http://www.math.uu.nl/people/jvoosten/seminar.html

Extra announcements

- Next friday, lecture by Sebastiaan Terwijn on the Medvedev Lattice
  See
  http://www.math.uu.nl/people/jvoosten/seminar.html
- 16:00 sharp at Wiskundegebouw, Room K11 (take stairs down)

Howework

- Confusion about exercise 2.2.11

Howework

- Confusion about exercise 2.2.11
- There is NO need to work within a formal system

Howework

- Confusion about exercise 2.2.11
- There is NO need to work within a formal system
- Confusion: $\exists\varphi\,\forall n$ versus $\forall n\,\exists\varphi$

Howework

- Confusion about exercise 2.2.11
- There is NO need to work within a formal system
- Confusion: $\exists\varphi\,\forall n$ versus $\forall n\,\exists\varphi$
- Bounded quantification up to p can not be done using p many disjunctions!

Howework

- Confusion about exercise 2.2.11
- There is NO need to work within a formal system
- Confusion: $\exists\varphi\,\forall n$ versus $\forall n\,\exists\varphi$
- Bounded quantification up to p can not be done using p many disjunctions!
- Students can go back to Daisuke with their homework to get a higher mark

Unlimited Register Machines

- Other computational models need to be introduced

Unlimited Register Machines

- Other computational models need to be introduced
- For sake of later applications

Unlimited Register Machines

- Other computational models need to be introduced
- For sake of later applications (and general education)

Unlimited Register Machines

- Other computational models need to be introduced
- For sake of later applications (and general education)
- URMs are also/better known as RAM: Random Access Machines

Unlimited Register Machines

- Other computational models need to be introduced
- For sake of later applications (and general education)
- URMs are also/better known as RAM: Random Access Machines
- Hardware: unbounded array of registers of unbounded capacity (starting at R_1)

Unlimited Register Machines

- Other computational models need to be introduced
- For sake of later applications (and general education)
- URMs are also/better known as RAM: Random Access Machines
- Hardware: unbounded array of registers of unbounded capacity (starting at R_1)
- Software: Four types of instructions

URMs

- Software: Four types of instructions

URMs

- Software: Four types of instructions
- Zero: $Z(n)$

URMs

- Software: Four types of instructions
- Zero: $Z(n)$
- Successor: $S(n)$

URMs

- Software: Four types of instructions
- Zero: $Z(n)$
- Successor: $S(n)$
- Transfer: $T(m, n)$: r_n becomes r_m

URMs

- Software: Four types of instructions
- Zero: $Z(n)$
- Successor: $S(n)$
- Transfer: $T(m, n)$: r_n becomes r_m
- Jump: $J(m, n, q)$ (If, then, else)

URMs

- Software: Four types of instructions
- Zero: $Z(n)$
- Successor: $S(n)$
- Transfer: $T(m, n)$: r_n becomes r_m
- Jump: $J(m, n, q)$ (If, then, else)
- Programs are numbered lists of instructions always executing the next instruction unless told otherwise by a Jump operation

URMs

- Software: Four types of instructions
- Zero: $Z(n)$
- Successor: $S(n)$
- Transfer: $T(m, n)$: r_n becomes r_m
- Jump: $J(m, n, q)$ (If, then, else)
- Programs are numbered lists of instructions always executing the next instruction unless told otherwise by a Jump operation
- A program stops if it enters a line with no program line on it

URM computable

- Input output convention

URM computable

- Input output convention
- Definition 2.3.2

URM computable

- Input output convention
- Definition 2.3.2
- URM Program P **computes** a function f

URM computable

- Input output convention
- Definition 2.3.2
- URM Program P **computes** a function f
- Function f is URM **computable**

URM computable

- Input output convention
- Definition 2.3.2 (totality can be relaxed)
- URM Program P **computes** a function f
- Function f is URM **computable**

URM computable

- Input output convention
- Definition 2.3.2 (totality can be relaxed)
- URM Program P **computes** a function f
- Function f is URM **computable**
- What is the class of URM computable functions?

URM computable

- Input output convention
- Definition 2.3.2 (totality can be relaxed)
- URM Program P **computes** a function f
- Function f is URM **computable**
- What is the class of URM computable functions?
- That is, find another – equivalent – characterization

URM computable functions

- Conjecture: URM Computable = Recursive

URM computable functions

- Conjecture: URM Computable = Recursive
- How to prove this?

URM computable functions

- Conjecture: URM Computable = Recursive
- How to prove this?
- Two parts!

URM computable functions

- Conjecture: URM Computable = Recursive
- How to prove this?
- Two parts!
- One part: By induction!

URM computable functions

- Conjecture: URM Computable = Recursive
- How to prove this?
- Two parts!
- One part: By induction!
- Basic case

URM computable functions

- Conjecture: URM Computable = Recursive
- How to prove this?
- Two parts!
- One part: By induction!
- Basic case
- and constructions

URM computable functions

- Closure under composition

URM computable functions

- Closure under composition
- Example: for one variable

URM computable functions

- Closure under composition
- Example: for one variable
- Intuitive idea: concatenate programs

URM computable functions

- Closure under composition
- Example: for one variable
- Intuitive idea: concatenate programs
- Three problems:

URM computable functions

- Closure under composition
- Example: for one variable
- Intuitive idea: concatenate programs
- Three problems:
- P_g may halt somewhere in the middle of P_f

URM computable functions

- Closure under composition
- Example: for one variable
- Intuitive idea: concatenate programs
- Three problems:
- P_g may halt somewhere in the middle of P_f
- Solution: assume programs are in *standard form*

URM computable functions

- Closure under composition
- Example: for one variable
- Intuitive idea: concatenate programs
- Three problems:
- P_g may halt somewhere in the middle of P_f
- Solution: assume programs are in *standard form*
- Problem/solution 2: renumber

URM computable functions

- Closure under composition
- Example: for one variable
- Intuitive idea: concatenate programs
- Three problems:
- P_g may halt somewhere in the middle of P_f
- Solution: assume programs are in *standard form*
- Problem/solution 2: renumber
- Problem 3: Input/Output convention: how many registers are used?

URM computable functions

- Proof by intimidation

URM computable functions

- Proof by intimidation
- Proof by hand waving

URM computable functions

- Proof by intimidation
- Proof by hand waving
- Thm: f is URM computable iff f is Recursive

URM computable functions

- Proof by intimidation
- Proof by hand waving
- Thm: f is URM computable iff f is Recursive
- By relaxing Definition 2.3.2 you can describe Partial Recursive

Turing machines: a classical paradigm

- Hardware: two sided infinite tape with reading/writing head

Turing machines: a classical paradigm

- Hardware: two sided infinite tape with reading/writing head
- Language: tape symbols, set of internal states, action symbols

Turing machines: a classical paradigm

- Hardware: two sided infinite tape with reading/writing head
- Language: tape symbols, set of internal states, action symbols
- Instructions are quadruples

Turing machines: a classical paradigm

- Hardware: two sided infinite tape with reading/writing head
- Language: tape symbols, set of internal states, action symbols
- Instructions are quadruples
- $Q = \langle q_i S A q_j \rangle$

Turing machines: a classical paradigm

- Hardware: two sided infinite tape with reading/writing head
- Language: tape symbols, set of internal states, action symbols
- Instructions are quadruples
- $Q = \langle q_i S A q_j \rangle$
- A program is a **consistent** set of instructions

Turing machines: a classical paradigm

- Hardware: two sided infinite tape with reading/writing head
- Language: tape symbols, set of internal states, action symbols
- Instructions are quadruples
- $Q = \langle q_i S A q_j \rangle$
- A program is a **consistent** set of instructions
- better name would have been “deterministic”

Turing machines: a classical paradigm

- Hardware: two sided infinite tape with reading/writing head
- Language: tape symbols, set of internal states, action symbols
- Instructions are quadruples
- $Q = \langle q_i S A q_j \rangle$
- A program is a **consistent** set of instructions
- better name would have been “deterministic”
- A program Halts if it runs out of instructions

Turing machines

- How to compute a function

Turing machines

- How to compute a function
- Input/output conventions

Turing machines

- How to compute a function
- Input/output conventions
- Input is unary (which is not a good choice)

Turing machines

- How to compute a function
- Input/output conventions
- Input is unary (which is not a good choice)
- Input: n is represented by $n+1$ consecutive 1's

Turing machines

- How to compute a function
- Input/output conventions
- Input is unary (which is not a good choice)
- Input: n is represented by $n + 1$ consecutive 1's
- Output: number of 1's on the tape

Turing machines

- How to compute a function
- Input/output conventions
- Input is unary (which is not a good choice)
- Input: n is represented by $n + 1$ consecutive 1's
- Output: number of 1's on the tape
- Again, define M computing a function, and a function being computable

Turing machines

- How to compute a function
- Input/output conventions
- Input is unary (which is not a good choice)
- Input: n is represented by $n + 1$ consecutive 1's
- Output: number of 1's on the tape
- Again, define M computing a function, and a function being computable
- Successor function becomes very easy!

Turing Machines

- Very true statement in Cooper:

Turing Machines

- Very true statement in Cooper:
- Composing Turing Programs is much more rewarding than reading them!

Turing Machines

- Very true statement in Cooper:
- Composing Turing Programs is much more rewarding than reading them!
- No Concrete Machines here

Turing Machines

- Very true statement in Cooper:
- Composing Turing Programs is much more rewarding than reading them!
- No Concrete Machines here
- How to deal with composition?

Turing Machines

- Very true statement in Cooper:
- Composing Turing Programs is much more rewarding than reading them!
- No Concrete Machines here
- How to deal with composition?
- Hard problem (?): Output becomes Input

Turing Machines

- Very true statement in Cooper:
- Composing Turing Programs is much more rewarding than reading them!
- No Concrete Machines here
- How to deal with composition?
- Hard problem (?): Output becomes Input
- Thm: Turing computable = Partial Recursive

Computations and theories

- We shall now make a link between computations and formal theories

Computations and theories

- We shall now make a link between computations and formal theories
- This yields: yet another characterization of the recursive functions

Computations and theories

- We shall now make a link between computations and formal theories
- This yields: yet another characterization of the recursive functions
- Plus the precursor to Gödel's first incompleteness theorem

Formal theories

- Specify three ingredients

Formal theories

- Specify three ingredients
- Language

Formal theories

- Specify three ingredients
- Language
- Axioms

Formal theories

- Specify three ingredients
- Language
- Axioms
- Rules

Formal theories

- Specify three ingredients
- Language
- Axioms
- Rules
- Model for idealized science

Peano Arithmetic

- Well known arithmetical first order system

Peano Arithmetic

- Well known arithmetical first order system
- Chapter 3

Peano Arithmetic

- Well known arithmetical first order system
- Chapter 3 (please send me corrections)

Peano Arithmetic

- Well known arithmetical first order system
- Chapter 3 (please send me corrections)
- Language consists of $\{+, {}', \times, 0, =\}$

Peano Arithmetic

- Well known arithmetical first order system
- Chapter 3 (please send me corrections)
- Language consists of $\{+, \,', \times, 0, =\}$
- Axioms, come in two groups

Peano Arithmetic

- Well known arithmetical first order system
- Chapter 3 (please send me corrections)
- Language consists of $\{+, \ ', \times, 0, =\}$
- Axioms, come in two groups
- The Logical axioms

Peano Arithmetic

- Well known arithmetical first order system
- Chapter 3 (please send me corrections)
- Language consists of $\{+, \;', \times, 0, =\}$
- Axioms, come in two groups
- The Logical axioms
- The non-logical axioms

Peano Arithmetic

- Well known arithmetical first order system
- Chapter 3 (please send me corrections)
- Language consists of $\{+, \ ', \times, 0, =\}$
- Axioms, come in two groups
- The Logical axioms
- The non-logical axioms
- The rules: (MP) and (GEN)

Models of Peano Arithmetic

- Note: there are infinitely many induction axioms (it is proved that finitely many can never be enough)

Models of Peano Arithmetic

- Note: there are infinitely many induction axioms (it is proved that finitely many can never be enough)
- What do models of PA look like

Models of Peano Arithmetic

- Note: there are infinitely many induction axioms (it is proved that finitely many can never be enough)
- What do models of PA look like
- Of course: standard model

Models of Peano Arithmetic

- Note: there are infinitely many induction axioms (it is proved that finitely many can never be enough)
- What do models of PA look like
- Of course: standard model
- Many more models

Some model constructions

- Gödel's completeness theorem: Any consistent first order theory has a model

Some model constructions

- Gödel's completeness theorem: Any consistent first order theory has a model
- The Compactness Theorem

Some model constructions

- Gödel's completeness theorem: Any consistent first order theory has a model
- The Compactness Theorem
- The (downwards) Löwenheim-Skolem Theorem

Some model constructions

- Gödel’s completeness theorem: Any consistent first order theory has a model
- The Compactness Theorem
- The (downwards) Löwenheim-Skolem Theorem
- What is countable?

What is countable?

- Countable: there is a surjection from the natural numbers

What is countable?

- Countable: there is a surjection from the natural numbers
- Examples: pairs

What is countable?

- Countable: there is a surjection from the natural numbers
- Examples: pairs
- Rationals

What is countable?

- Countable: there is a surjection from the natural numbers
- Examples: pairs
- Rationals
- Finite sequences of natural numbers

What is countable?

- Countable: there is a surjection from the natural numbers
- Examples: pairs
- Rationals
- Finite sequences of natural numbers
- Are there uncountable sets?

What is countable?

- Countable: there is a surjection from the natural numbers
- Examples: pairs
- Rationals
- Finite sequences of natural numbers
- Are there uncountable sets?
- Cantor: Yes, eg $[0, 1]$

Models of PA

- There exists a non-standard model of PA

Models of PA

- There exists a non-standard model of PA
- What do these non-standard models look like?

Models of PA

- There exists a non-standard model of PA
- What do these non-standard models look like?
- Order structure is unique

Models of PA

- There exists a non-standard model of PA
- What do these non-standard models look like?
- Order structure is unique
- However (Friedman) uncountably (continuous) many

PA and the representable functions

- What does it mean to represent a function in PA?

PA and the representable functions

- What does it mean to represent a function in PA?
- Reduce it to relations

PA and the representable functions

- What does it mean to represent a function in PA?
- Reduce it to relations
- Which functions are representable in PA?

PA and the representable functions

- What does it mean to represent a function in PA?
- Reduce it to relations
- Which functions are representable in PA?
- Theorem: all recursive functions are representable in PA

PA and the representable functions

- What does it mean to represent a function in PA?
- Reduce it to relations
- Which functions are representable in PA?
- Theorem: all recursive functions are representable in PA
- Later we shall prove the reverse

PA and the representable functions

- What does it mean to represent a function in PA?
- Reduce it to relations
- Which functions are representable in PA?
- Theorem: all recursive functions are representable in PA
- Later we shall prove the reverse
- Thus, we have yet another characterization of the recursive functions