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`

Questions and remarks

- Enrollment (regular) possible till September 25

Questions and remarks

- Enrollment (regular) possible till September 25
- After that, probably also possible, but harder

Questions and remarks

- Enrollment (regular) possible till September 25
- After that, probably also possible, but harder
- Poblems/questions: contact Tanja Kassenaar and me.

Non primitive recursive functions

- Loads of them

Non primitive recursive functions

- Loads of them
- Best example: the Ackermann function

Non primitive recursive functions

- Loads of them
- Best example: the Ackermann function
- $A(m, 0) = m + 1$

Non primitive recursive functions

- Loads of them
- Best example: the Ackermann function
- $A(m, 0) = m + 1$
- $A(m, 1) = m + 2$

Non primitive recursive functions

- Loads of them
- Best example: the Ackermann function
- $A(m, 0) = m + 1$
- $A(m, 1) = m + 2$
- $A(m, 2) = 2 \times m + 3$

Non primitive recursive functions

- Loads of them
- Best example: the Ackermann function
- $A(m, 0) = m + 1$
- $A(m, 1) = m + 2$
- $A(m, 2) = 2 \times m + 3$
- $A(m, 3) = 2^{m+3} - 3$

Non primitive recursive functions

- Loads of them
- Best example: the Ackermann function
- $A(m, 0) = m + 1$
- $A(m, 1) = m + 2$
- $A(m, 2) = 2 \times m + 3$
- $A(m, 3) = 2^{m+3} - 3$
- Each next step iterates over the previous one

The Ackermann function

- Play around a bit with it

The Ackermann function

- Play around a bit with it
- More fast growing than any function in PRIM

The Ackermann function

- Play around a bit with it
- More fast growing than any function in PRIM
- However, it is somehow computable

The Ackermann function

- Play around a bit with it
- More fast growing than any function in PRIM
- However, it is somehow computable
- Thus, something was missing

The Ackermann function

- Play around a bit with it
- More fast growing than any function in PRIM
- However, it is somehow computable
- Thus, something was missing
- Minimalisation

Minimalisation

- Defined as a search query

Minimalisation

- Defined as a search query
- We loose totality

Minimalisation

- Defined as a search query
- We loose totality
- Functions in computability are a bit different than in maths in that they don't need to be total

Minimalisation

- Defined as a search query
- We loose totality
- Functions in computability are a bit different than in maths in that they don't need to be total
- Is not allowed to jump over previous undefined values!

Minimalisation

- Defined as a search query
- We loose totality
- Functions in computability are a bit different than in maths in that they don't need to be total
- Is not allowed to jump over previous undefined values!
- Zero plays no essential role

Minimalisation

- Defined as a search query
- We loose totality
- Functions in computability are a bit different than in maths in that they don't need to be total
- Is not allowed to jump over previous undefined values!
- Zero plays no essential role
- funny terminology: total partial

Minimalisation

- Defined as a search query
- We loose totality
- Functions in computability are a bit different than in maths in that they don't need to be total
- Is not allowed to jump over previous undefined values!
- Zero plays no essential role
- funny terminology: total partial
- Conjecture: this is all there is in a sense we shall specify a bit more in a minute

Minimalisation

- Defined as a search query
- We loose totality
- Functions in computability are a bit different than in maths in that they don't need to be total
- Is not allowed to jump over previous undefined values!
- Zero plays no essential role
- funny terminology: total partial
- Conjecture: this is all there is in a sense we shall specify a bit more in a minute

Recursive functions: an aside

- And what about Ackermann?

Recursive functions: an aside

- And what about Ackermann?
- We need coding tricks!

Recursive functions: an aside

- And what about Ackermann?
- We need coding tricks!
- MDRP-Theorem: one application of minimalisation in the end suffices

Recursive functions: an aside

- And what about Ackermann?
- We need coding tricks!
- MDRP-Theorem: one application of minimalisation in the end suffices (In theory)

Church's Thesis (CT thesis)

- Effectively computable coincides with p.r.

Church's Thesis (CT thesis)

- Effectively computable coincides with p.r. (What is said here?)
- (Total and Eff. comp.) coincides with recursive

Church's Thesis (CT thesis)

- Effectively computable coincides with p.r. (What is said here?)
- (Total and Eff. comp.) coincides with recursive
- Eff. comp. in some intuitive sense

Church's Thesis (CT thesis)

- Effectively computable coincides with p.r. (What is said here?)
- (Total and Eff. comp.) coincides with recursive
- Eff. comp. in some intuitive sense
- Long holding thesis

Church's Thesis (CT thesis)

- Effectively computable coincides with p.r. (What is said here?)
- (Total and Eff. comp.) coincides with recursive
- Eff. comp. in some intuitive sense
- Long holding thesis
- We shall see some more protocols of computing

Church's Thesis (CT thesis)

- Effectively computable coincides with p.r. (What is said here?)
- (Total and Eff. comp.) coincides with recursive
- Eff. comp. in some intuitive sense
- Long holding thesis
- We shall see some more protocols of computing
- How to use CTT in practice?

Church's Thesis (CT thesis)

- Effectively computable coincides with p.r. (What is said here?)
- (Total and Eff. comp.) coincides with recursive
- Eff. comp. in some intuitive sense
- Long holding thesis
- We shall see some more protocols of computing
- How to use CTT in practice?
- Very, very useful

Church's Thesis (CT thesis)

- Effectively computable coincides with p.r. (What is said here?)
- (Total and Eff. comp.) coincides with recursive
- Eff. comp. in some intuitive sense
- Long holding thesis
- We shall see some more protocols of computing
- How to use CTT in practice?
- Very, very useful
- There is no equally undisputed thesis for primitive recursive around

Recursive relations

- Relations R and their characteristic functions χ_R

Recursive relations

- Relations R and their characteristic functions χ_R
- E.g., $\chi_<(m,n) = \mathsf{sg}(m \dot{-} n)$

Recursive relations

- Relations R and their characteristic functions χ_R
- E.g., $\chi_<(m,n) = \mathsf{sg}(m \dot{-} n)$
- Homework: $\chi_=$ on $\mathbb{N}$ is PRIM

Recursive relations

- Relations R and their characteristic functions χ_R
- E.g., $\chi_<(m,n) = \mathsf{sg}(m \dot{-} n)$
- Homework: $\chi_=$ on $\mathbb{N}$ is PRIM
- Again, we are going to build up a repertoire

Recursive relations

- Relations R and their characteristic functions χ_R
- E.g., $\chi_<(m, n) = \mathsf{sg}(m \dot{-} n)$
- Homework: $\chi_=$ on $\mathbb{N}$ is PRIM
- Again, we are going to build up a repertoire
- Number of divisors is PRIM ($\chi_{m|n}$, sg, and bounded sum)

Recursive relations

- Relations R and their characteristic functions χ_R
- E.g., $\chi_<(m, n) = \mathsf{sg}(m \dot{-} n)$
- Homework: $\chi_=$ on $\mathbb{N}$ is PRIM
- Again, we are going to build up a repertoire
- Number of divisors is PRIM ($\chi_{m|n}$, sg, and bounded sum)
- Important: closure properties

Closure properties

- Closed under the Boolean connectives

Closure properties

- Closed under the Boolean connectives
- Theorem: every finite set is in PRIM

Closure properties

- Closed under the Boolean connectives
- Theorem: every finite set is in PRIM
- Closure under bounded quantification

Closure properties

- Closed under the Boolean connectives
- Theorem: every finite set is in PRIM
- Closure under bounded quantification
- And a lot more!

Coding techniques

- Important for coding: p_n

Coding techniques

- Important for coding: p_n
- $p_0 = 2$

Coding techniques

- Important for coding: p_n
- $p_0 = 2$
- $p_{n+1} = \mu z[z > p_n \wedge \mathsf{Pr}(z)]$

Coding techniques

- Important for coding: p_n
- $p_0 = 2$
- $p_{n+1} = \mu z[z > p_n \wedge \mathsf{Pr}(z)]$ (can we have them PRIM ?)
- There is a recursive pairing function

Coding techniques

- Important for coding: p_n
- $p_0 = 2$
- $p_{n+1} = \mu z[z > p_n \wedge \mathsf{Pr}(z)]$ (can we have them PRIM ?)
- There is a recursive pairing function
- With recursive inverses π_i

Coding techniques

- Important for coding: p_n
- $p_0 = 2$
- $p_{n+1} = \mu z[z > p_n \wedge \mathsf{Pr}(z)]$ (can we have them PRIM ?)
- There is a PRIM recursive pairing function
- With PRIM recursive inverses π_i
- We can even code finite arbitrary sequences in a PRIM way!

Coding techniques

- Important for coding: p_n
- $p_0 = 2$
- $p_{n+1} = \mu z[z > p_n \wedge \mathsf{Pr}(z)]$ (can we have them PRIM ?)
- There is a PRIM recursive pairing function
- With PRIM recursive inverses π_i
- We can even code finite arbitrary sequences in a PRIM way!
- $(m)_i$ can be used for (de)coding strings of arbitrary length: $\langle a_0, \ldots, a_n \rangle \mapsto p_0^n \cdot p_1^{a_0} \cdot \ldots \cdot p_{n+1}^{a_n}$

Coding techniques

- Important for coding: p_n
- $p_0 = 2$
- $p_{n+1} = \mu z[z > p_n \wedge \mathsf{Pr}(z)]$ (can we have them PRIM ?)
- There is a PRIM recursive pairing function
- With PRIM recursive inverses π_i
- We can even code finite arbitrary sequences in a PRIM way!
- $(m)_i$ can be used for (de)coding strings of arbitrary length: $\langle a_0, \ldots, a_n \rangle \mapsto p_0^n \cdot p_1^{a_0} \cdot \ldots \cdot p_{n+1}^{a_n}$
- Important first application of coding techniques: Course-of-Values Recursion