# $\lambda$-calculus

Simona Ronchi Della Rocca

Università degli Studi di Torino

summer school “Logic and Computation”

Goettingen, July 24-30, 2016

# A bit of history

- Alonzo Church (1936) The $\lambda$-calculus as formal account of computation. Proof of the undecidability of the ALT problem.
- John Mc Carthy (1962) The LISP programming language, inspired to $\lambda$-calculus. LISP is a list-processing language with a function-abstraction facility. Used mostly for applications in artificial intelligence.
- Peter J. Landin (1964) A mechanical evaluation of ISWIM (acronym of "If you See What I Mean"), a $\lambda$-calculus enriched by some constants for numerals, through a machine named SECD (acronym of "Stack, Environment, Code, Dump").
  Translation of ALGOL 60 into $\lambda$-calculus.

- Corrado Bohm (1966) The CUCH language, mixing $\lambda$-calculus and combinatory logic.

  (1968) The separability theorem. The first "semantical" result about $\lambda$-calculus: if two irreducible terms of $\lambda$-calculus are syntactically different then they are also semantically different, so they need to have different interpretation in any model.

- Dana Scott (1976) The first mathematical model of $\lambda$-calculus, based on a solution of the equation: $D = [D \to D]$, where $[D \to D]$ represents the class of continuous functions from $D$ to $D$. Birth of the denotational semantics.

- Roger Hindley, Robin Milner (1968 - 1978) Typed $\lambda$-calculus and ML programming language with automatic type inference.

- Gordon Plotkin (1975) The call-by-value version of $\lambda$-calculus. In fact the $\lambda$-calculus uses a call-by-name parameter passing style.
  (1981) Structural operational semantics of $\lambda$-calculus.
- Henk Barendregt (1981) A complete compendium about $\lambda$-calculus.
- Jean Yves Girard and the french school (1971) The second order $\lambda$-calculus.
  (1987 - 2016) The Linear Logic and a quantitative interpretation of $\lambda$-calculus.
- Coppo, Dezani, RDR and Torino school (1984 - 2016) The logical description of semantical properties of $\lambda$-calculus.

An historical view of $\lambda$-calculus:

Felice Cardone and J. Roger Hindley: "Lambda-calculus and Combinators in the 20th Century", chapter of "The Handbook of the History of Logic" (edited by D. Gabbay and J. Woods), Vol.5:533-627, Elsevier, 2008.

# The language $\Lambda$

Let Var be a countable set of variables. The set $\Lambda$ of $\lambda$-terms is inductively defined as follows:

- $x \in$ Var implies $x \in \Lambda$ (variable)
- $M \in \Lambda$ and $x \in$ Var implies $(\lambda x.M) \in \Lambda$ (abstraction)
- $M \in \Lambda$ and $N \in \Lambda$ implies $(MN) \in \Lambda$ (application)
- $\equiv$ denotes the syntactical identity on terms.

Abbreviations:

- $\lambda x_1 ... x_n.M$ denotes $(\lambda x_1(...(\lambda x_n.M)...))$
- $MN_1N_2...N_n$ denotes $(...((MN_1)N_2)...N_n)$.

## Example

$\lambda x.xx$, $\lambda x.x(\lambda z.zy)$, $\lambda y.(\lambda x.x)(\lambda uv.u)$

$I \equiv \lambda x.x, K \equiv \lambda xy.x, O \equiv \lambda xy.y, D \equiv \lambda x.xx, E \equiv \lambda xy.xy.$

# Free and bound variables

The symbol $\lambda$ plays the role of binder for variables!

- The set of free variables of a term $M$, denoted by FV($M$), is inductively defined as follows:
  - $M \equiv x$ implies $\mathrm{FV}(M) = \{x\}$;
  - $M \equiv \lambda x.M'$ implies $\mathrm{FV}(M) = \mathrm{FV}(M') - \{x\}$;
  - $M \equiv PQ$ implies $\mathrm{FV}(M) = \mathrm{FV}(P) \cup \mathrm{FV}(Q)$.

  A variable is bound in $M$ if it is not free in $M$. BV($M$) denotes the set of bound variables of $M$.
- A term $M$ is closed if and only if $\mathrm{FV}(M) = \emptyset$. A term is open if it is not closed.
- Let $\Theta \subseteq \Lambda$, $\Theta^0$ is the restriction of $\Theta$ to closed terms.

**Example**

$\mathrm{FV}(\lambda z.(\lambda x.x(\lambda z.zy))(\lambda xyz.yz)) = \{y\}$, $\mathrm{FV}(\lambda z.x(\lambda x.xy)) = \{x, y\}$,

$\mathrm{FV}((\lambda yx.x)y) = \{y\}$.

# $\alpha$-reduction





The name of a bound variable is meaningless!

- $\lambda x.M \to_\alpha \lambda y.M[y/x]$ if $y$ does not occur in $M$.
- $=_\alpha$ is the reflexive, symmetric, transitive and contextual closure of $\to_\alpha$.

## Example

$\lambda x.x =_\alpha \lambda y.y =_\alpha \lambda z.z$, $\lambda xy.x =_\alpha \lambda xz.x$ and $\lambda xy.x =_\alpha \lambda yx.y$.

But:

$\lambda x.y \neq_\alpha \lambda x.x$ and $\lambda x.yx \neq_\alpha \lambda y.yy$.

$=$ denotes $\equiv \cup =_\alpha$. Terms will be considered modulo $=$.

# Substitution

Replacing the occurrences of a free variable $x$ in $M$ by a term $N$ (notation $M[N/x]$) can produce an incorrect result (capture of variables).

## Example

$(\lambda x.\lambda y.xy)[yz/x]$ produces $\lambda y.yzy$: the free occurrence of $y$ became bound!

## Example

Variables convention Assume that, for every term $M$ occurring in a certain context (definition, proof,...) $\mathrm{FV}(M) \cap \mathrm{BV}(M) = \emptyset$.

Thanks to this convention, the operation $M[N/x]$ can be simply defined as:

- $x[N/x] = N$
- $y[N/x] = y$, if $x \neq y$
- $(\lambda y.M)[N/x] = \lambda y.M[N/x]$
- $(PQ)[N/x] = (P[N/x]Q[N/x])$

# Context

Let Var be a countable set of variables, and [.] be a constant (the hole).

- The set $\Lambda_C$ of contexts is inductively defined as follows:
  - $[.] \in \Lambda_C$;
  - $x \in$ Var implies $x \in \Lambda_C$;
  - $C[.] \in \Lambda_C$ and $x \in$ Var imply $(\lambda x.C[.]) \in \Lambda_C$;
  - $C_1[.] \in \Lambda_C$ and $C_2[.] \in \Lambda_C$ imply $(C_1[.]C_2[.]) \in \Lambda_C$.

  Contexts will be denoted by $C[.], C'[.], C_1[.]$....
- Let $C[.]$ be a context and $M$ be a term. Then $C[M]$ denotes the term obtained by replacing by $M$ every occurrence of [.] in $C[.]$.

**Note**

Filling a hole in a context is not a substitution!

In fact free variables in $M$ can become bound in $C[M]$. For example, filling the hole of $\lambda x.[.]$ with the free variable $x$ gives as result the term $\lambda x.x$.

# $\beta$-reduction





The $\lambda$-calculus has just one reduction rule: the $\beta$-reduction.

- The $\beta$-reduction ($\rightarrow_\beta$) is the contextual closure of the following rule:

$$(\lambda x.M)N \rightarrow M[N/x].$$

  $(\lambda x.M)N$ is called a $\beta$-redex (or simply redex) and $M[N/x]$ is called its $\beta$-contractum (or simply contractum).

- $\rightarrow_\beta^*$ and $=_\beta$ are respectively the reflexive and transitive closure of $\rightarrow_\beta$ and the symmetric, reflexive and transitive closure of $\rightarrow_\beta$.

## Example

$(\lambda x.yxx)(\lambda z.z) \rightarrow_\beta y(\lambda z.z)(\lambda z.z)$

$\lambda x.y((\lambda z.z)(\lambda w.w)) \rightarrow_\beta \lambda x.y(\lambda z.z)$

# Normal form

- The general shape of a term is:

$$\lambda x_1 ... x_n . \zeta M_1 ... M_m \quad (n, m \geq 0)$$

  where $M_i$ are the arguments of $M$ $(1 \leq i \leq m)$ and $\zeta$ is the *head* of $M$.

- $\zeta$ is either a variable (head variable) or a redex (head redex)
- A term is in $\beta$-normal form (shortly normal form) if it does not contain occurrences of $\beta$-redexes. The general shape of a normal form is:

$$\lambda x_1 ... x_n . z M_1 ... M_m \quad (n, m \geq 0)$$

  where $M_i$ is a normal form $(1 \leq i \leq m)$.

- A term has a normal form if it reduces to a normal form.
- Let $NF$ be the set of all normal forms.

# Confluence

The natural interpretation of a term $\lambda x.M$ is a function whose formal parameter is $x$. The interpretation of $(\lambda x.M)N$ is the application of the function $\lambda x.M$ to the actual parameter $N$ and so the $\beta$-reduction rule models the replacement of the formal parameter $x$ by the actual parameter $N$ in the body $M$ of the function. Thus the $\beta$-normal form of a term, if it exists, can be seen as the final result of a computation.
The following fundamental theorem implies that this interpretation is correct, i.e. if the computation process stops, then the result is unique.

## Theorem (Confluence)

*Let $M \to_\beta^* N_1$ and $M \to_\beta^* N_2$.*
*There is $Q$ such that both $N_1 \to_\beta^* Q$ and $N_2 \to_\beta^* Q$.*

# Proof (sketch)

(Taït and Martin Löf, simplified by Takahashi)

- The deterministic parallel reduction $\hookrightarrow$ is inductively defined as follows:
  - $x \hookrightarrow x$
  - $M \hookrightarrow N$ implies $\lambda x.M \hookrightarrow \lambda x.N$
  - $M \hookrightarrow M', N \hookrightarrow N'$ imply $MN \hookrightarrow M'N'$
  - $M \hookrightarrow M', N \hookrightarrow N'$ imply $(\lambda x.M)N \hookrightarrow M'[N'/x]$
- The non-deterministic parallel reduction $\Rightarrow$ is inductively defined as follows:
  - $x \Rightarrow x$
  - $M \Rightarrow N$ implies $\lambda x.M \Rightarrow \lambda x.N$
  - $M \Rightarrow M', N \Rightarrow N'$ imply $MN \Rightarrow M'N'$
  - $M \Rightarrow M', N \Rightarrow N'$ imply $(\lambda x.M)N \Rightarrow M'[N'/x]$
  - $M \Rightarrow M', N \Rightarrow N'$ imply $(\lambda x.M)N \Rightarrow (\lambda x.M')N'$

Roughly speaking, the deterministic parallel reduction reduces in one step all the redexes of a term, while the non-deterministic one reduces a subset of them.

## Lemma (Properties of parallel reductions)

- $M \to_\beta N$ *implies* $M \Rightarrow N$
- $M \Rightarrow N$ *implies* $M \to_\beta^* N$
- $\to_\beta^*$ *is the transitive closure of* $\Rightarrow$

For every term $M$, there is a unique term $N$ such that $M \hookrightarrow N$ ($N$ is called the complete development of $M$, and is denoted by $[M]$).

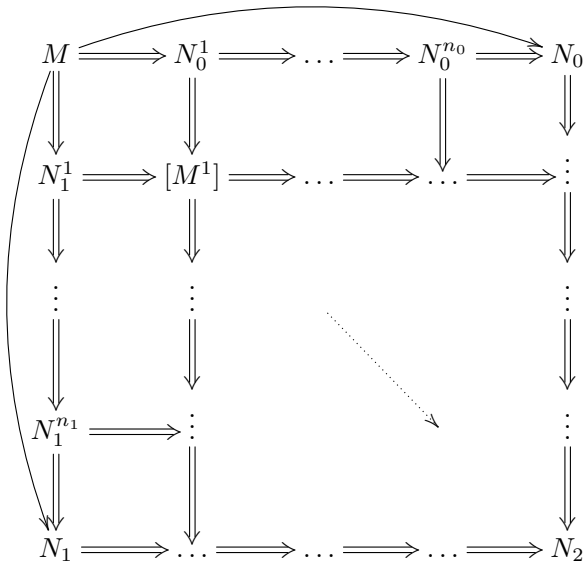
## Property

$M \hookrightarrow P$ and $M \hookrightarrow Q$ implies $P = Q$.

## Lemma (Diamond Property of $\Rightarrow$)

*If* $M \Rightarrow N_0$ *and* $M \Rightarrow N_1$ *then there is* $N_2$ *such that both* $N_0 \Rightarrow N_2$ *and* $N_1 \Rightarrow N_2$.

$\rightarrow_\beta^*$ is the transitive closure of $\Rightarrow$. So there are $N_0^1, ..., N_0^{n_0}$, $N_1^1, ..., N_1^{n_1}$ $(n_0, n_1 \geq 1)$ such that $M \Rightarrow N_0^1 ... \Rightarrow N_0^{n_0} \Rightarrow N_0$ and $M \Rightarrow N_1^1 ... \Rightarrow N_m^{n_1} \Rightarrow N_1$. Then the proof follows by applying repeatedly the diamond property of $\Rightarrow$ (diamond closure)




**Diamond Closure.**

# Unicity





## Corollary

*The normal form of a term, if it exists, is unique.*

*Proof.* Assume by absurdum that a term $M$ has two different normal forms $M_1$ and $M_2$. Then, by the Confluence Theorem, there is a term $N$ such that both $M_1$ and $M_2$ $\beta$-reduce to $N$, against the hypothesis that both are normal forms. □

## Example

Terms without normal form:

$DD \rightarrow_\beta DD \rightarrow_\beta ....DD....$

$\lambda x.(\lambda y.x(yy))(\lambda y.x(yy)) \rightarrow_\beta \lambda x.x((\lambda y.x(yy))(\lambda y.x(yy))) \rightarrow_\beta$

$\lambda x.x(x((\lambda y.x(yy))(\lambda y.x(yy)))...$

# Sequentialisation



## Example

$DD \to_\beta DD \to_\beta ....DD....$

$KI(DD) \to_\beta KI(DD) \to_\beta ... \to_\beta KI(DD) \to_\beta ...$

(choosing at every step the innermost redex)

$KI(DD) \to_\beta I$

(choosing the leftmost redex).

Let us impose a total order between redexes.

- the degree of a redex in a term $M$ is the number of $\lambda$'s precedings it in reading $M$ from left to right.

## Example

$M = \lambda x.((\lambda z.z)x)((\lambda zw.Iwz)D)$

The degree of $(\lambda z.z)x$ is $1$.

The degree of $(\lambda zw.Iwz)D$ is $2$.

The degree of $Iw$ is $3$.

# Standardization

- A sequence of reductions $M_0 \to_\beta M_1 \to_\beta ..... \to_\beta M_n$ is standard if the degree of the redex contracted in $M_i$ is less than the degree of the redex contracted in $M_{i+1}$, for every $i < n$.

## Example

$M = \lambda x.((\lambda z.z)x)((\lambda zw.Iwz)D)$

$M \to_\beta \lambda x.((\lambda z.z)x)(\lambda w.IwD) \to_\beta \lambda x.((\lambda z.z)x)(\lambda w.wD)$ is standard.

$M \to_\beta \lambda x.((\lambda z.z)x)((\lambda zw.Iwz)D) \to_\beta \lambda x.((\lambda z.z)x)(\lambda w.IwD) \to_\beta$

$\lambda x.x(\lambda w.IwD)$ is not standard.

## Theorem (Standardization)

*$M \to_\beta^* N$ implies there is a standard reduction sequence from $M$ to $N$.*

# Reaching the normal form





## Note

The redex with the minimum degree is the leftmost redex.

## Corollary

*A term reduces to its normal form (if it exists) by reducing, at every step, the leftmost redex.*

## Example

The redex with minimum degree in $KI(DD)$ is $KI$. So reducing it we have:

$KI(DD) \rightarrow_\beta I$

# Solvability

- A term $M$ is solvable if there is a sequence of terms $N_0, ..., N_m$ such that

$$(\lambda x_0 ... x_n . M) N_0 ... N_m \rightarrow_\beta^* I$$

$(\{x_0, ..., x_n\} = \mathsf{FV}(M))$

- A term is unsolvable if it is not solvable.

Informally speaking, a solvable term is a term in some sense computationally meaningful. In fact, let $M \in \Lambda^0$ be solvable, and let $P \in \Lambda$: we can always find a sequence $\vec{N}$ of terms such that $M\vec{N}$ reduces to $P$: just take the sequence $\vec{Q}$ such that $M\vec{Q} \rightarrow_\beta^* I$, which exists since $M$ is solvable, and pose $\vec{N} \equiv \vec{Q}P$. So a closed solvable term can mimic the behaviour of any term, if applied to suitable arguments.

# Head normal form

- A term is in head normal form (hnf) if its head is a variable
- A term has head normal form if it reduces to a hnf.
- Let $HNF$ be the set of all terms in head normal form.

## Example

Every normal form is a hnf.

$\lambda x.x(DD)$ is in hnf, but not in normal form.

$\lambda x.Ix(DD) \to_\beta \lambda x.x(DD)$, so it has hnf, but it has not normal form.

$DD$ has neither hnf nor normal form.

## NOTE

The head normal form of a term is not unique!

Let $M = \lambda x.Ix(II)$. $M \to_\beta \lambda x.x(II) \to_\beta \lambda x.xI$.

Both $\lambda x.x(II)$ and $\lambda x.xI$ are hnf's!

# Head normal form

Let $M \equiv \lambda x_1 \ldots x_n.zM_1 \ldots M_m$.

- $n$ is the order of $M$
- $m$ is the degree of $M$

## Property

Let $M$ have hnf. Then there are unique $n, m$ such that, for every $N$, $M \to_\beta^* N$ where $N$ in hnf implies that the order and degree of $N$ are respectively $n$ and $m$.

*Proof.* By contraposition, let $M$ have two hnf's, with different order and degree, i.e., $M \to_\beta^* P_1 = \lambda x_1 ... x_n.xM_1 ... M_m$ and $M \to_\beta^* P_2 = \lambda x_1 ... x_p.xN_1 ... N_q$, where $n \neq p$ and/or $m \neq q$. By the confluence theorem, there must be a term $Q$ such that both $P_1 \to_\beta^* Q$ and $P_2 \to_\beta^* Q$. But this impossible, since the only redexes can occur in $M_i$ or in $N_j$, and their reduction cannot change any of $n, m, p, q$ $(1 \leq i \leq m, 1 \leq j \leq q)$ . □

# Fixed point theorem

A term having head-normal-form with an infinite behavior:

$$\text{Let } R = (\lambda y.x(yy))(\lambda y.x(yy)) \text{ and } Y = \lambda x.R$$

$$Y \to_\beta \lambda x.xR \to_\beta^* \lambda x.\underbrace{x(x(...(x}_{n} R \underbrace{)...))}_{n} \qquad \text{for every } n \geq 0$$

**Theorem (Fixed point operator)**

*Every term $M \in \Lambda$ has a fixed point, i.e., for every term $M$ there is a term $N$ such that $MN =_\beta N$.*

*Proof.* It is immediate to check that, for every $M$, $YM =_\beta M(YM)$. Hence $YM$ is a fixed point of $M$. □

The fixed point theorem is the key property for proving that the $\lambda$-calculus has the computational power of the partial computable functions.
It corresponds to recursion in programming languages.

# Solvability theorem





Head normal forms supply a syntactical account of solvability!

## Theorem (Solvability)

*A term is solvable if and only if it has a head normal form.*

The proof is based on the following property.

## Property

i) The lack of hnf is preserved by substitution, i.e. if $M$ hasn't hnf then $M[N/y]$ hasn't hnf too, for all $y \in$ Var.

ii) The lack of hnf is preserved by head contexts, i.e. if $M$ hasn't hnf then $(\lambda\vec{x}.M)\vec{N}$ hasn't hnf too, for all $\vec{x}$ and $\vec{N}$.

*Proof.* Exercise!!! □

# Proof of Solvability theorem





## proof

($\Leftarrow$) Without loss of generality, we can assume that $M$ is closed. Let $M = \lambda x_1 ... x_n . x_i M_1 ... M_m$ $(1 \leq i \leq n)$. Let $P_i = \lambda x_1 ... x_{m+1} . x_{m+1}$. Then for every sequence $P_1 ... P_i ... P_n$, where $P_j$ is any term, for $i \neq j$,

$$M P_1 ... P_i ... P_n =_\beta I.$$

($\Rightarrow$) If $M$ hasn't hnf, then by Property, for all head context $C[.]$, $C[M]$ hasn't hnf; in particular, $C[M]$ can't be reduced to $I$.

# Theories

In order to model the computation, $=_\beta$ is too weak.
For example, if we want to model the termination property, both the terms $DD$ and $(\lambda x.xxx)(\lambda x.xxx)$ represent running forever programs,
while the two terms are $\neq_\beta$ each other.
Indeed $DD \to_\beta DD$ and $(\lambda x.xxx)(\lambda x.xxx) \to_\beta (\lambda x.xxx)(\lambda x.xxx)(\lambda x.xxx)$.
So it would be natural to consider them equal in this particular setting.
But if we want to take into account not only termination, but also the size of terms, they need to be different, in fact the first one reduces to itself
while the second increases its size during the reduction.
As we will see in the sequel, all interesting interpretations of the calculus equate also terms that are not $=$.

- $\mathcal{T} \subseteq \Lambda \times \Lambda$ is a congruence if and only if $(M, N) \in \mathcal{T}$ implies $(C[M], C[N]) \in \mathcal{T}$, for all contect $C[.]$.
- $\mathcal{T} \subseteq \Lambda \times \Lambda$ is a $\lambda$-theory if and only if it is a congruence and $M = N$ implies $(M, N) \in \mathcal{T}$.

# Properties of theories

- A $\lambda$-theory $\mathcal{T}$ is consistent if and only if there are $M, N \in \Lambda$ such that $M \neq_{\mathcal{T}} N$. Otherwise $\mathcal{T}$ is inconsistent.
- A $\lambda$-theory $\mathcal{T}$ is maximal if and only if it has no consistent extension, i.e., for all $M, N \in \Lambda$, such that $M \neq_{\mathcal{T}} N$, any $\lambda$-theory $\mathcal{T}'$ containing $\mathcal{T}$ and such that $M =_{\mathcal{T}'} N$ is inconsistent.
- A $\lambda$-theory is sensible if it equates all unsolvable terms;
- A $\lambda$-theory is semi-sensible if it never equates a solvable and an unsolvable term.

## Example

The theory $\mathcal{T}_\beta = \{(M, N) \mid M =_\beta N\}$ is consistent, sensible and semi-sensible.

The theory $\mathcal{T}_\beta \cup (I, DD)$ is consistent, not sensible and not semi-sensible.

# Extensionality

A theory is extensional if all terms in it (not only abstractions) have a functional behaviour. So, in an extensional theory $\mathcal{T}$, the equality between terms must be extensional (in the usual sense), i.e., it must satisfy the property:

$$\text{(EXT)}\ Mx =_{\mathcal{T}} Nx \Rightarrow M =_{\mathcal{T}} N \qquad x \notin \mathsf{FV}(M) \cup \mathsf{FV}(N).$$

Clearly $=_\beta$ does not satisfy (EXT).
In fact, (EXT) holds for $=_\beta$ only if it is restricted to terms which reduce to an abstraction: indeed $xy =_\beta (\lambda z.xz)y$, but $x \neq_\beta \lambda z.xz$.

# $\eta$-reduction

The least extensional extension of $=_\beta$ is induced by the $\eta$-reduction rule, defined as follows.

- The $\eta$-reduction ($\to_\eta$) is the contextual closure of the following rule:
  $\lambda x.Mx \to_\eta M$ if $x \notin FV(M)$;
  $\lambda x.Mx$ is a $\eta$-redex and $M$ is its contractum;
- $M \to_{\beta\eta} N$ if $N$ is obtained from $M$ by reducing either a $\beta$ or a $\eta$ redex in $M$;
- $\to^*_{\beta\eta}$ and $=_{\beta\eta}$ are respectively the reflexive and transitive closure of $\to_{\beta\eta}$ and the symmetric, reflexive and transitive closure of $\to_{\beta\eta}$.

**Theorem**

*$=_{\beta\eta}$ is the least extensional extension of $=_\beta$.*

*Let $\mathcal{T}$ be a theory such that $I =_{\mathcal{T}} E$. Then $\mathcal{T}$ is extensional.*

*Proof.* Exercise! □

# Evaluation

The evaluation of a term consists of applying to it a sequence of reduction rules until a result is given. The most natural notion of result is a normal form, but we can define other reasonable notions.

- A set of results is any set $\Theta \subseteq \Lambda$ such that:
  - $\Theta$ is closed under $\rightarrow_\beta$;
  - if $M =_\beta N$ and $N \in \Theta$ then there is $P \in \Theta$ such that $M \rightarrow_\beta^* P$.

The first condition of the previous definition takes into account the fact that a result represents the output of an evaluation, so, also in case it can be further reduced, it cannot become an unevaluated term.
The second condition simply comes from the fact that $\rightarrow_\beta$ is our evaluation rule.
Notice that normal forms are results, according to the definition.

## Example

$HNF$ is a set of results.

The set of weak head normal forms, i.e., the set

$\{\lambda x.M \mid M \in \Lambda\} \cup \{xM_0...M_n \mid x \in \text{Var}, M_i \in \Lambda\}$ is a set of results.

# Evaluation relation





- An evaluation relation $\mathbf{O}$ on the $\lambda$-calculus with respect to a set of results $\Theta$ (notation: $\mathbf{O} \in \mathcal{E}(\Theta)$) is any subset of $\Lambda \times \Theta$, such that $(M, N) \in \mathbf{O}$ implies $M \rightarrow_\beta^* N$.

An evaluation relation can be presented by using a formal system.

A *logical rule*, or briefly rule, has the following shape:

$$\frac{\mathfrak{P}_1 \ \ldots\ldots \ \mathfrak{P}_m}{\mathfrak{C}} \ \text{name}$$

where the premises $\mathfrak{P}_i$ $(1 \leq i \leq m)$ and the conclusion $\mathfrak{C}$ are logical judgments (written using meta-variables); while name is the name of the rule.

The intended meaning of a rule is that, for every instance $s$ of the meta-variables in the rule, $s(\mathfrak{C})$ is implied by the logical AND of $s(\mathfrak{P}_i)$ $(1 \leq i \leq m)$.

- A derivation is a finite tree of logical rules, such that each leaf is an axiom, each intermediate node has as premises the consequences of its son nodes and its consequence is one of the premises of its father node. The conclusion of the root node is the proved judgment.
- A formal system defining an evaluation relation $\mathbf{O} \in \mathcal{E}(\Theta)$ is a set of logical rules for establishing judgments of the shape $M \Downarrow_{\mathbf{O}} N$, whose meaning is $(M, N) \in \mathbf{O}$.
- A formal system establishing judgments of the shape $M \Downarrow_{\mathbf{O}} N$ can be viewed as a logical representation of a reduction machine; in particular the evaluation process of the machine is simulated by a derivation in the logical system.
  - $M \Downarrow_{\mathbf{O}} N$ means that on input $M$, the reduction machine $\mathbf{O}$ stops and gives as output N
  - $M \Downarrow_{\mathbf{O}}$ means that on input $M$, the reduction machine $\mathbf{O}$ stops
  - $M \Uparrow_{\mathbf{O}}$ means on input $M$, the reduction machine never stops

# Operational semantics



An evaluation relation $\mathbf{O} \in \mathcal{E}(\Theta)$ induces naturally an operational semantics, i.e. a pre-order relation (and an equivalence relation) on terms.

- $M \preceq_{\mathbf{O}} N \quad \Leftrightarrow \quad \forall C[.].C[M], C[N] \in \Lambda^0 \ (C[M] \Downarrow_{\mathbf{O}} \Rightarrow C[N] \Downarrow_{\mathbf{O}})$.
- $M \approx_{\mathbf{O}} N \quad \Leftrightarrow \quad M \preceq_{\mathbf{O}} N$ and $N \preceq_{\mathbf{O}} M$.

This operational equivalence amounts to Leibniz Equality Principle for programs, i.e., a criterion for establishing equivalence on the basis of the behaviour of programs regarded as black boxes. It is natural to model a program by a closed term. So a context can be viewed as a partially specified program, where every occurrence of the hole denotes a place that must be filled by a subprogram, while a generic term can be viewed as a subprogram.
So two terms are equivalent if they can be replaced by each other in the same program without changing its behaviour (with respect to an evaluation relation $\mathbf{O}$).
Since we are considering pure calculi, i.e., calculi without constants, the only behaviour we can observe on terms is the termination, and this justify the previous definition of operational semantics.

# The $\mathbf{H}$-abstract machine





- Take as set of results the set of head normal forms.
- $\mathbf{H} \in \mathcal{E}(\Lambda\text{-HNF})$ is the evaluation relation induced by the abstract machine consisting of the following rules:

$$\frac{m \geq 0}{xM_1 \ldots M_m \Downarrow_{\mathbf{H}} xM_1 \ldots M_m} \ (var)$$

$$\frac{M \Downarrow_{\mathbf{H}} N}{\lambda x.M \Downarrow_{\mathbf{H}} \lambda x.N} \ (abs)$$

$$\frac{P[Q/x]M_1 \ldots M_m \Downarrow_{\mathbf{H}} N}{(\lambda x.P)QM_1 \ldots M_m \Downarrow_{\mathbf{H}} N} \ (head)$$

# Examples

- $\lambda x.(\lambda uv.xuv)I(DD) \Downarrow_{\mathbf{H}} \lambda x.xI(DD)$.

  In fact we can build the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{xI(DD) \Downarrow_{\mathbf{H}} xI(DD)}\,(var)}{(\lambda v.xIv)(DD) \Downarrow_{\mathbf{H}} xI(DD)}\,(head)}{(\lambda uv.xuv)I(DD) \Downarrow_{\mathbf{H}} xI(DD)}\,(head)}{\lambda x.(\lambda uv.xuv)I(DD) \Downarrow_{\mathbf{H}} \lambda x.xI(DD)}\,(abs)$$

  Note that, in the particular case of the system $\Downarrow_{\mathbf{H}}$, every derivation is such that each node has a unique son.

- $DD \Uparrow_{\mathbf{H}}$

  (Exercize!) Proof hint: Assume by absurdum $DD \Downarrow_{\mathbf{H}}$, take the smallest (w.r.t. the number of rules) derivation proving it, and prove that it implies the existence of a smaller one.

# $H$-characterization

The system $\Downarrow_{\mathbf{H}}$ characterizes completely the class of terms having head normal form.

## Theorem ($H$-characterization)

- *$M \Downarrow_{\mathbf{H}}$ if and only if $M$ has hnf.*

*Proof.*

($\Rightarrow$) By induction on the definition of $\Downarrow_{\mathbf{H}}$.

($\Leftarrow$) $M$ has hnf means that there is $N \in HNF$ such that $M \to_\beta^* N \in HNF$. By standardization, at every step the leftmost redex is reduced. The proof is done by induction on the length of the reduction sequence $M \to_\beta^* M'$. Let $M = \lambda x_1 \ldots x_n.\zeta M_1 \ldots M_m$ $(n, m \in \mathbb{N})$.

If $\zeta$ is a variable then $M$ is already in hnf. In fact $M \Downarrow_{\mathbf{H}} M$, by $n$ applications of rule *(abs)* and one application of the rule *(var)*.

If $\zeta \equiv (\lambda x.P)Q$ then by induction, $P[Q/x]M_1 \ldots M_m \Downarrow_{\mathbf{H}} N$, for some $N$; thus $M \Downarrow_{\mathbf{H}} \lambda x_1 \ldots x_n.N$, by $n$ applications of rule *(abs)* and one application of the rule *(head)*.

# H-operational semantics

- $M \preceq_{\mathbf{H}} N$ if and only if, for all context $C[.]$ such that $C[M], C[N] \in \Lambda^0$, ($C[M] \Downarrow_{\mathbf{H}}$ implies $C[N] \Downarrow_{\mathbf{H}}$ ).
- $M \approx_{\mathbf{H}} N$ if and only if $M \preceq_{\mathbf{H}} N$ and $N \preceq_{\mathbf{H}} M$.

## Property

$I \approx_{\mathbf{H}} E$.

*Proof.* hint Let assume by absurdum that the two terms are different. This means that there is a context $C[.]$ discriminating them. Let $C[.]$ be such that $C[I] \Downarrow_{\mathbf{H}}$ while $C[E] \Uparrow_{\mathbf{H}}$. Let $C[.]$ be a minimal discriminating context for $I$ and $E$, and prove that this implies the existence of a smaller discriminating context. The case $C[I] \Uparrow_{\mathbf{H}}$ and $C[E] \Downarrow_{\mathbf{H}}$ is symmetric.

□

# H-operational semantics





## Corollary

*The* $\mathbf{H}$*-theory is extensional.*

The next theorem proves an unexpected result.
It has been proved that the $\mathbf{H}$-operational semantics is extensional,
i.e., it is closed under $\eta$-equality.
It can be proved that it equates also terms that can
be obtained each other by means of an infinite number of $\eta$-reductions.

Let $E_\infty \equiv Y(\lambda xyz.y(xz))$ where $Y$ is the fixed point operator. For every $M$,

$$E_\infty M =_\beta \lambda z.M(E_\infty z) =_\beta \lambda z.M(\lambda z_1.z(E_\infty z_1)) =_\beta \lambda z.M(\lambda z_1.z(\lambda z_2.z_1(E_\infty z_2))),$$

and so on. So $z$ can be viewed as obtained from $E_\infty z$ by means of an infinite number of application of $\eta$ -reduction rule.

## Theorem

$I \approx_{\mathbf{H}} E_\infty$

*Proof.* Through a denotational model. □

# The N-abstract machine

- Take as set of results the set $NF$ of normal forms.
- $\mathbf{N} \in \mathcal{E}(\Lambda\text{-NF})$ is the evaluation relation induced by the formal system proving judgments of the shape

$$M \Downarrow_{\mathbf{N}} N$$

where $M \in \Lambda$ and $N \in \Lambda$-NF. It consists of the following rules:

$$\frac{(M_i \Downarrow_{\mathbf{N}} N_i)_{(i \leq m)}}{xM_1 \ldots M_m \Downarrow_{\mathbf{N}} xN_1 \ldots N_m}\ (var)$$

$$\frac{M \Downarrow_{\mathbf{N}} N}{\lambda x.M \Downarrow_{\mathbf{N}} \lambda x.N}\ (abs)$$

$$\frac{P[Q/x]M_1 \ldots M_m \Downarrow_{\mathbf{N}} N}{(\lambda x.P)QM_1 \ldots M_m \Downarrow_{\mathbf{N}} N}\ (head)$$

## Example

$\lambda x_1 x_2.x_1(ID)((\lambda uv.u)(II)x_2) \Downarrow_{\mathbf{N}} \lambda x_1 x_2.x_1 DI$, as shown by the following derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x \Downarrow_{\mathbf{N}} x}(var)}{xx \Downarrow_{\mathbf{N}} xx}(var)}{\lambda x.xx \Downarrow_{\mathbf{N}} \lambda x.xx}(abs)}{ID \Downarrow_{\mathbf{N}} D}(head) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x \Downarrow_{\mathbf{N}} x}(var)}{\lambda x.x \Downarrow_{\mathbf{N}} \lambda x.x}(abs)}{II \Downarrow_{\mathbf{N}} I}(head)}{\lambda v.II \Downarrow_{\mathbf{N}} \lambda v.I}(abs)}{(\lambda uv.u)(II) \Downarrow_{\mathbf{N}} \lambda v.I}(head)}{x_1(ID)((\lambda uv.u)(II)) \Downarrow_{\mathbf{N}} x_1 D(\lambda v.I)}(var)$$

$$\dfrac{x_1(ID)((\lambda uv.u)(II)) \Downarrow_{\mathbf{N}} x_1 D(\lambda v.I)}{\lambda x_2.x_1(ID)((\lambda uv.u)(II)) \Downarrow_{\mathbf{N}} \lambda x_2.x_1 D(\lambda v.I)}(abs)$$

$$\dfrac{\lambda x_2.x_1(ID)((\lambda uv.u)(II)) \Downarrow_{\mathbf{N}} \lambda x_2.x_1 D(\lambda v.I)}{\lambda x_1 x_2.x_1(ID)((\lambda uv.u)(II)) \Downarrow_{\mathbf{N}} \lambda x_1 x_2.x_1 D(\lambda v.I)}(abs)$$

# N-characterization

The system $\Downarrow_{\mathbf{N}}$ characterizes the class of $\beta$-normal forms.

## Theorem

*$M \Downarrow_{\mathbf{N}}$ if and only if $M$ has nf.*

*Proof.*

- $\Rightarrow$ By induction on the definition of $\Downarrow_{\mathbf{N}}$.
- $\Leftarrow$ If $M \to_{\Lambda}^{*} N \in$NF then $M \to_{\beta}^{*} N$. The proof follows by induction on the pair $(M, p)$, where $p$ is the length of the reduction sequence $M \to_{\beta}^{*} N$, ordered in a lexicographic way. By standardization, at every step the leftmost redex is reduced.

  Let $M \equiv \lambda x_1 \ldots x_n.\zeta M_1 \ldots M_m$.

  If $\zeta$ is a variable then $N \equiv \lambda x_1 \ldots x_n.\zeta nf(M_1) \ldots nf(M_m)$. By induction $M_i \Downarrow_{\mathbf{N}}$ $(1 \leq i \leq m)$, thus $M \Downarrow_{\mathbf{N}}$ by rule *(var)* having as premises the derivation proving $M_i \Downarrow_{\mathbf{N}}$ and $n$ instances of *(abs)*.

  If $\zeta \equiv (\lambda x.P)Q$ then $\Downarrow_{\mathbf{N}} (M) \equiv \lambda x_1 \ldots x_n. \Downarrow_{\mathbf{N}} (P[Q/x]M_1 \ldots M_m)$; so, by induction, $P[Q/x]M_1 \ldots M_m \Downarrow_{\mathbf{N}} R$, for some $R$; hence $(\lambda x.P)QM_1 \ldots M_m \Downarrow_{\mathbf{N}} N$, by applying rule *(head)* and $M \Downarrow_{\mathbf{N}} \lambda x_1 \ldots x_n.N$ by $n$ instances of *(abs)*.

$\square$

# $\mathbf{N}$ operational semantics

- $M \preceq_{\mathbf{N}} N$ if and only if, for all context $C[.]$ such that $C[M], C[N] \in \Lambda^0$, ($C[M] \Downarrow_{\mathbf{N}}$ implies $C[N] \Downarrow_{\mathbf{N}}$ ).
- $M \approx_{\mathbf{N}} N$ if and only if $M \preceq_{\mathbf{N}} N$ and $N \preceq_{\mathbf{N}} M$.

## Property

$I \approx_{\mathbf{N}} E$.

## Corollary

*The theory $\approx_{\mathbf{N}}$ is extensional.*

But the theory $\approx_{\mathbf{N}}$ is able to grasp only finite number of $\eta$-reductions, differently from $\approx_{\mathbf{H}}$

## Theorem

$I \not\approx_{\mathbf{N}} E_\infty$

# The $\lambda$ calculus as programming language

The $\lambda$-calculus can be seen as paradigm for programming languages.
In fact it has the computational power of Turing machines, or, equivalently, it is computationally complete. The completeness can be achieved without adding special constants to the language, but all data structures needed for computing, in particular booleans, natural numbers and functions, can be coded into $\Lambda$.
Both the reduction machines that are been presented can be effectively used for computing. Here the machine $\Downarrow_{\mathbf{H}}$ will be used.
In order to prove the computational completeness, we will refer to the class of recursive funtions introduced by Kleene.

# Representing data structures: booleans





## Definition

Let $\mathbf{O} \in \mathcal{E}(\Theta)$ be an evaluation relation.

An **O-representation of booleans** is any set $\{T, F\}$ such that:

- $T, F \in \Theta$;
- there is a term $Cond$ such that, for every $M, N \in \Theta$:

$$Cond\ TMN \Downarrow_{\mathbf{O}} M \qquad Cond\ FMN \Downarrow_{\mathbf{O}} N.$$

## Example

In $\mathbf{H}$-operational semantics, choose:

$$\mathsf{T} = \lambda xy.x \text{ and } \mathsf{F} = \lambda xy.y.$$

In this cases $Cond$ can be taken as the identity term $I$, or simply omitted.

In fact, if $M, N \in HNF$ then $\mathsf{T}MN \Downarrow_{\mathbf{H}} M$ and $\mathsf{F}MN \Downarrow_{\mathbf{H}} N$.

# Representing data structures: pairs





## Definition

Let $\mathbf{O} \in \mathcal{E}(\Theta)$ be an evaluation relation.

Let $M, N \in \Theta$. $[M, N]$ is a pair if the are terms $P_1, P_2$ such that:

- $P_1[M, N] \Downarrow_{\mathbf{O}} M$
- $P_2[M, N] \Downarrow_{\mathbf{O}} N$

## Example

In $\mathbf{H}$-operational semantics, choose:

$$P_1 = \lambda x.x\mathsf{T} \text{ and } P_2 = \lambda x.x\mathsf{F}.$$

# Representing data structures: numerals

## Definition (Inspired to Peano's definition of natural numbers)

Let $\mathbf{O} \in \mathcal{E}(\Theta)$ be an evaluation relation.

A **$\mathbf{O}$-numeral system** is a 5-tuple $\langle \mathbb{B}, Zero, Succ, Test, Pred \rangle$, where:

- $\mathbb{B}$ is an $\mathbf{O}$-representation of booleans
- $Zero, Succ, Test, Pred \in \Theta$ are such that, for all $n \in \mathbb{N}$:
  - $\underbrace{Succ\ (\ ...\ (Succ}_{n}\ Zero)...) \Downarrow_{\mathbf{O}}$. Let $\underbrace{Succ\ (\ ...\ (Succ}_{n}\ Zero)...) \Downarrow_{\mathbf{O}} \ulcorner n \urcorner$ and let $\ulcorner n \urcorner \in \Theta$. $\ulcorner n \urcorner$ is the *numeral representation* of $n$;
  - $P \Downarrow_{\mathbf{O}} \ulcorner n \urcorner$ implies $Succ\ P \Downarrow_{\mathbf{O}} \ulcorner n+1 \urcorner$;
  - $P \Downarrow_{\mathbf{O}} Zero$ and $Q \Downarrow_{\mathbf{O}} \ulcorner n+1 \urcorner$ imply $Test\ P \Downarrow_{\mathbf{O}} T$ and $Test\ Q \Downarrow_{\mathbf{O}} F$;
  - $P \Downarrow_{\mathbf{O}} \ulcorner n+1 \urcorner$ implies $Pred\ P \Downarrow_{\mathbf{O}} \ulcorner n \urcorner$.

# Representing data structures: numerals





## Example

In $\mathbf{H}$-operational semantics, choose:

- $\mathbb{B} = \{\mathsf{T}, \mathsf{F}\}$
- Zero $= [\mathsf{T}, \mathsf{T}]$;
- Succ $= \lambda t.t(\lambda uvx.x\mathsf{F}(\lambda y.yuv))$;
- Test $= \lambda x.x\mathsf{T}$;
- Pred $= \lambda x.x\mathsf{F}$.

## Exercise

Verify that:

- $\ulcorner n \urcorner \equiv \underbrace{[\mathsf{F}, [\mathsf{F}....[\mathsf{F}}_{n}, \mathsf{Zero}]...]]$
- $\ulcorner n+1 \urcorner \equiv \lambda x.x\mathsf{F}\ulcorner n \urcorner$

# Representing functions





## Definition

Let $\mathbf{O} \in \mathcal{E}(\Theta)$ be an evaluation relation, and and let $\phi$ be a partial recursive function with arity $p \in \mathbb{N}$; let $\ulcorner n \urcorner$ be the numeral representation of $n \in \mathbb{N}$ in an $\mathbf{O}$-numeral system.

- $\phi$ is $\mathbf{O}$-representable if and only if there is a term $\ulcorner \phi \urcorner \in \Lambda^0$ such that, for all terms $N_i$ such that $N_i \Downarrow_{\mathbf{O}} \ulcorner n_i \urcorner$ $(1 \leq i \leq p; n_1, ..., n_p \in \mathbb{N})$:
  - if $\phi(n_1, ..., n_p)$ is defined then $\ulcorner \phi \urcorner N_1 ... N_p \Downarrow_{\mathbf{O}} \ulcorner \phi(n_1, ..., n_p) \urcorner$;
  - if $\phi(n_1, ..., n_p)$ is undefined then $\ulcorner \phi \urcorner N_1 ... N_p \Uparrow_{\mathbf{O}}$.

# Kleene's primitive recursive functions





## Definition (Primitive Recursive Functions (shortly **PRF**))

- Basis functions:
  1. The zero function $Z(n) = 0$
  2. The successor function: $S(n) = n + 1$
  3. The projection functions $\pi^m_i(x_1, ..., x_m) = x_i \quad (1 \leq i \leq m \in \mathbb{N})$.

## Example

In **H**-operational semantics, choose:

- $\ulcorner Z \urcorner \equiv \lambda x.\mathsf{Zero}$;
- $\ulcorner S \urcorner \equiv \mathsf{Succ}$;
- $\ulcorner \pi^i_m \urcorner \equiv \lambda x_1 \ldots x_m.x_i \ (1 \leq i \leq m \in \mathbb{N})$.

## Definition

- Composition:

$h : \mathbb{N}^n \to \mathbb{N} \in \mathbf{PRF}$ and $g_1,\ldots,g_n : \mathbb{N}^m \to \mathbb{N} \in \mathbf{PRF}$ imply

$$f(x_1,\ldots,x_m) = h(g_1(x_1,\ldots,x_m),\ldots..,g_n(x_1,\ldots,x_m)) \in \mathbf{PRF} \quad (\mathbf{n},\mathbf{m} \in \mathbb{N})$$

In order to represent the composition of partial functions,
the main problem is to make the representation "strict";
namely, when a function is applied to an undefined argument
then its evaluation must diverge.
The proposed solution takes into account the fact that terms
representing natural numbers are in head normal form and so solvable.

## Lemma

*If* $M \Downarrow_{\mathbf{H}} \ulcorner n \urcorner$ *then* $MKII \Downarrow_{\mathbf{H}} I$.

## Example

In **H**-operational semantics, choose:

$$F = \lambda x_1\ldots x_m.\ulcorner h \urcorner(\ulcorner g_1 \urcorner x_1\ldots x_m).....(\ulcorner g_n \urcorner x_1\ldots x_m) \qquad \text{and}$$

$$\ulcorner f \urcorner = \lambda x_1\ldots x_m.(\ulcorner g_1 \urcorner x_1\ldots x_m KII).....(\ulcorner g_n \urcorner x_1\ldots x_m KII)(Fx_1\ldots x_m).$$

## Definition

- Primitive recursion:

$h : \mathbb{N}^{m+2} \to \mathbb{N} \in \mathbf{PRF}$ and $g : \mathbb{N}^m \to \mathbb{N} \in \mathbf{PRF}$, then $f \in \mathbf{PRF}$:

$$f(k, x_1, \ldots, x_m) = \begin{cases} g(x_1, \ldots, x_m) & \text{if } k = 0 \\ h(f(k-1, x_1, \ldots, x_m), k-1, x_1, \ldots, x_m) & \text{otherwise.} \end{cases}$$

In order to represent the functions built by primitive recursion and by minimalization, a fixed point operator is needed.
We already proved that $Y$ plays the role of a fixed point.
But, while $YM =_\beta M(YM)$, it does not hold that
$YM \to_\beta^* M(YM)$, which is a necessary condition
for using it as recursion operator in a reduction machine.

## Theorem

*Let* $Y_{\mathbf{H}} = (\lambda xy.y(xxy))(\lambda xy.y(xxy))$. *If* $M \in \Lambda$ *then* $Y_{\mathbf{H}}M \to_\beta^* M(Y_{\mathbf{H}}M)$;
*moreover* $Y_{\mathbf{H}}M \Downarrow_{\mathbf{H}} R$ *if and only if* $M(Y_{\mathbf{H}}M) \Downarrow_{\mathbf{H}} R$.

## Example

$\ulcorner f \urcorner$ is $\mathbf{H}$-represented by $Y_{\mathbf{H}} P$, where $P$ is:

$$\lambda t y x_1 \ldots x_m.\text{Test } y(\ulcorner g \urcorner x_1 \ldots x_m)(\ulcorner h \urcorner(t(\text{Pred } y)x_1 \ldots x_m(\text{Pred } y)x_1 \ldots x_m)).$$

*Proof.* [hint] Let $N_i \Downarrow_{\mathbf{O}} \ulcorner n_i \urcorner$, $Q \Downarrow_{\mathbf{O}} \ulcorner k \urcorner$ for some $k, n_i \in \mathbb{N}$ $(1 \leq i \leq m)$; the proof can be given by induction on $k$. □

# Kleene's recursive functions





## Definition (Minimalization)

Let $h : \mathbb{N}^2 \to \mathbb{N}$ be a total function and let $x \in \mathbb{N}$. Then a function $f : \mathbb{N} \to \mathbb{N}$ can be defined by minimalization from $h$ in the following way:

$$f(x) = \mu y\,[h(x,y) = 0] = \begin{cases} \min\{k \in \mathbb{N} \mid h(x,k) = 0\} & \text{if such a } k \in \mathbb{N} \text{ exists} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

## Example

Let $P \equiv \lambda thxy.(\text{Test}(hxy))y(thx(\text{Succ } y))$.

Let $h : \mathbb{N}^2 \to \mathbb{N}$ be an $\mathbf{H}$-representable partial recursive function. Let $N$ and $Q$ be such that $N \Downarrow_{\mathbf{O}} \ulcorner n \urcorner$ and $Q \Downarrow_{\mathbf{H}} \ulcorner k \urcorner$.

- If $h(n,k) = 0$ then $(Y_{\mathbf{H}}P)\ulcorner h \urcorner NQ \Downarrow_{\mathbf{H}} \ulcorner k \urcorner$.
- If $h(n,k) \neq 0$ then:

$$(Y_{\mathbf{H}}P)\ulcorner h \urcorner NQ \Downarrow_{\mathbf{H}} R \text{ if and only if } (Y_{\mathbf{H}}P)\ulcorner h \urcorner N(\text{Succ } Q) \Downarrow_{\mathbf{H}} R.$$

# Computational completeness





**Definition (Recursive Functions (shortly RF))**

A function $f : \mathbb{N}^m \to \mathbb{N}$ ($m \in \mathbb{N}$) is *recursive* if and only if one of the following conditions holds:

- $f$ is a primitive recursive function;
- $f$ is defined by composition of partial recursive functions;
- $f$ is defined by minimalization starting from a total recursive function.

**Theorem**

*The $\lambda$-calculus has the computational power of the Kleene recursive functions, so of the Turing machines.*