Typy proste

15 kwietnia 2013

Simple types

Types:

- Type constant $\mathbf{0}$ is a type.
- If σ and τ are types then $(\sigma \to \tau)$ is a type.

Alternatywne podejście: inne stałe lub zmienne typowe.

Konwencja:

- Zamiast $(\tau \to (\sigma \to \rho))$ piszemy $\tau \to \sigma \to \rho$.

Każdy typ ma postać $\tau_1 \to \cdots \to \tau_n \to \mathit{atom}$.

Church style syntax (orthodox)

Assume infinite sets V_τ of variables of each type τ.

Define sets T_τ of terms of type τ:

- A variable of type τ is a term of type τ;
- If $M \in T_{\sigma\to\tau}$ and $N \in T_\sigma$ then $(MN) \in T_\tau$;
- If $M \in T_\tau$ and $x \in V_\sigma$ then $(\lambda x\, M) \in T_{\sigma\to\tau}$.

Write M^σ for $M \in T_\sigma$ and define beta-reduction by

$$(\lambda x^\sigma .\, M^\tau)N^\sigma \Rightarrow M[x^\sigma := N].$$

Church and Curry

Church style:

- New syntax, built-in types.
- Every term has exactly one type.
- No "untypable" terms.

Curry style:

- Ordinary untyped lambda-terms.
- Types are derivable properties of terms.
- System of type assignment rules.
- A term may have many types or none.
- Typability not obvious.

Non-orthodox Church

Type-assignment with type annotations on bound variables.

$$\Gamma\,(x:\sigma) \vdash x : \sigma \;\; \text{(Var)}$$

$$\frac{\Gamma(x:\sigma) \vdash M : \tau}{\Gamma \vdash \lambda x{:}\sigma\, M : \sigma \to \tau} \;\; \text{(Abs)}$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \;\; \text{(App)}$$

Fact: If $\Gamma \vdash M : \tau$ and $\Gamma \vdash M : \sigma$ then $\tau = \sigma$.

Relating systems

Orthodox Church terms are like

- Non-orthodox terms in a fixed infinite environment.
- Curry-style type derivations.

Konwencja: Typy jako górne indeksy, np.

$$(\lambda x^{\sigma}\, M^{\tau})N^{\sigma} : \tau$$

Properties

Subject reduction property:
Beta-eta reduction preserves types.

Strong normalization:
Every typed term is strongly normalizing.

Definable functions

Liczebniki Churcha $\mathbf{n} = \lambda fx.\, f^n(x)$ mają każdy typ postaci

$$\omega_\sigma = (\sigma \to \sigma) \to (\sigma \to \sigma).$$

A function $f : \mathbb{N}^k \to \mathbb{N}$ is β-*definable in type* ω_σ if there is a closed term F such that

- $\vdash F : \omega_\sigma \to \cdots \to \omega_\sigma \to \omega_\sigma$;
- If $f(n_1, \ldots, n_k) = m$ then $F\mathbf{n}_1 \ldots \mathbf{n}_k =_\beta \mathbf{m}$.

Examples

- Addition: $\lambda n^{\omega_\sigma} \lambda m^{\omega_\sigma} \lambda f^{\sigma \to \sigma} \lambda x^{\sigma} . \, nf(mfx)$;

Examples

- Addition: $\lambda n^{\omega_\sigma} \lambda m^{\omega_\sigma} \lambda f^{\sigma \to \sigma} \lambda x^{\sigma}.\, nf(mfx)$;
- Multiplication: $\lambda n^{\omega_\sigma} \lambda m^{\omega_\sigma} \lambda f^{\sigma \to \sigma} \lambda x^{\sigma}.\, n(mf)x$;

Examples

- Addition: $\lambda n^{\omega_\sigma} \lambda m^{\omega_\sigma} \lambda f^{\sigma \to \sigma} \lambda x^\sigma .\, nf(mfx)$;
- Multiplication: $\lambda n^{\omega_\sigma} \lambda m^{\omega_\sigma} \lambda f^{\sigma \to \sigma} \lambda x^\sigma .\, n(mf)x$;
- Test for zero ($\mathit{if}\ n = 0\ \mathit{then}\ m\ \mathit{else}\ k$):

  $\lambda n^{\omega_\sigma} \lambda m^{\omega_\sigma} \lambda k^{\omega_\sigma} \lambda f^{\sigma \to \sigma} \lambda x^\sigma .\, n(\lambda y^\sigma . kfx)(mfx)$.

Extended polynomials (wielomiany warunkowe)

The least class of functions containing:

- Addition;
- Multiplication;
- Test for zero;
- Constants zero and one;
- Projections,

and closed under compositions.

Example: $f(x, y) = \texttt{if}\ x = 0\ \texttt{then}\ \texttt{if}\ y = 0\ \texttt{then}\ p_1(x, y)$ $\texttt{else}\ p_2(x, y)\ \texttt{else}\ \texttt{if}\ y = 0\ \texttt{then}\ p_3(x, y)\ \texttt{else}\ p_4(x, y)$.

Definable functions

Theorem (H. Schwichtenberg'76):

For every σ the functions beta-definable in type ω_σ are exactly the extended polynomials.

$\circ$

More definable functions

A function f is *non-uniformly* definable if there is a closed term F such that

- $\vdash F : \omega_{\sigma_1} \rightarrow \cdots \rightarrow \omega_{\sigma_k} \rightarrow \omega_{\sigma}$;
- If $f(n_1, \ldots, n_k) = m$ then $F\mathbf{n}_1 \ldots \mathbf{n}_k =_\beta \mathbf{m}$.

More definable functions

A function f is *non-uniformly* definable if there is a closed term F such that

- $\vdash F : \omega_{\sigma_1} \rightarrow \cdots \rightarrow \omega_{\sigma_k} \rightarrow \omega_{\sigma}$;
- If $f(n_1, \ldots, n_k) = m$ then $F\mathbf{n}_1 \ldots \mathbf{n}_k =_\beta \mathbf{m}$.

Examples:

- The predecessor function $p(n) = n \dot{-} 1$ and the exponentiation function $exp(m, n) = m^n$ are non-uniformly definable. (Easy)

More definable functions

A function f is *non-uniformly* definable if there is a closed term F such that

- $\vdash F : \omega_{\sigma_1} \to \cdots \to \omega_{\sigma_k} \to \omega_\sigma$;
- If $f(n_1, \ldots, n_k) = m$ then $F\mathbf{n}_1 \ldots \mathbf{n}_k =_\beta \mathbf{m}$.

Examples:

- The predecessor function $p(n) = n \dot{-} 1$ and the exponentiation function $exp(m, n) = m^n$ are non-uniformly definable. (Easy)
- The subtraction $minus(m, n) = m \dot{-} n$ and equality test $Eq(m, n) =$`if` $m = n$ `then 0 else 1` are not definable non-uniformly. (Hard)

Equality

Theorem (R. Statman'79): *The equality problem*

Are two well-typed terms beta-equal?

is non-elementary. That is, for no fixed k it is solvable in time

$$\left.2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}\right\}k$$

Equality

Theorem (R. Statman'79): *The equality problem*

Are two well-typed terms beta-equal?

is non-elementary. That is, for no fixed k it is solvable in time

$$\left.2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}\right\}k$$

Exercise: How long is the normal form of $\mathbf{2}\cdots\mathbf{2}xy$?

○

The inhabitation problem

Inhabitation problem:

Given Γ, τ, is there M such that $\Gamma \vdash M : \tau$?

The inhabitation problem

Inhabitation problem:

Given Γ, τ, is there M such that $\Gamma \vdash M : \tau$?

Fact (R. Statman):

Inhabitation in simple types is decidable and Pspace*-complete.*

○

Representing data types

- Natural numbers are generated by
  - Constant $0 : \mathbf{int}$;
  - Successor $s : \mathbf{int} \rightarrow \mathbf{int}$.

Representing data types

- Natural numbers are generated by
  - Constant $0 : \mathbf{int}$;
  - Successor $s : \mathbf{int} \to \mathbf{int}$.

  They correspond to long normal forms of type

$$\omega = (\mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$$

Representing data types

- Natural numbers are generated by
  - Constant $0 : \mathbf{int}$;
  - Successor $s : \mathbf{int} \to \mathbf{int}$.

  They correspond to long normal forms of type

  $$\omega = (\mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$$

- Words over $\{a, b\}$ are generated by
  - Constant $\varepsilon : \mathbf{word}$;
  - Two successors $\lambda\!\!\lambda w\,(a \cdot w)$ and $\lambda\!\!\lambda w\,(b \cdot w)$ of type $\mathbf{word} \to \mathbf{word}$.

Representing data types

- Natural numbers are generated by
  - Constant $0 : \mathbf{int}$;
  - Successor $s : \mathbf{int} \to \mathbf{int}$.

  They correspond to long normal forms of type
  $$\omega = (\mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$$
- Words over $\{a, b\}$ are generated by
  - Constant $\varepsilon : \mathbf{word}$;
  - Two successors $\lambda\!\!\lambda w\,(a \cdot w)$ and $\lambda\!\!\lambda w\,(b \cdot w)$ of type $\mathbf{word} \to \mathbf{word}$.

  They correspond to long normal forms of type
  $$\mathbf{word} = (\mathbf{0} \to \mathbf{0}) \to (\mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$$

Representing data types

- Binary trees are generated by
  - Constant $nil : \mathbf{tree}$;
  - Constructor $cons : \mathbf{tree} \to \mathbf{tree} \to \mathbf{tree}$.

Representing data types

- Binary trees are generated by
  - Constant *nil* : **tree**;
  - Constructor *cons* : **tree** $\to$ **tree** $\to$ **tree**.

  They correspond to long normal forms of type

$$\mathbf{tree} = (\mathbf{0} \to \mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$$

Representing data types

- Binary trees are generated by
  - Constant *nil* : **tree**;
  - Constructor *cons* : **tree** $\to$ **tree** $\to$ **tree**.

  They correspond to long normal forms of type

$$\mathbf{tree} = (\mathbf{0} \to \mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$$

Generalization:
Free algebras correspond to types of order two, i.e, of the form

$$(\mathbf{0}^{n_1} \to \mathbf{0}) \to \cdots \to (\mathbf{0}^{n_k} \to \mathbf{0}) \to \mathbf{0}$$

○

Type reducibility

Definition: Type τ is *reducible* to type σ iff there exists a closed term $\Phi : \tau \to \sigma$ such that the operator $\lambda\!\!\lambda M{:}\tau.\, \Phi M$ is injective on closed terms, i.e.,

$$\Phi M_1 =_{\beta\eta} \Phi M_2 \quad \text{implies} \quad M_1 =_{\beta\eta} M_2$$

for closed $M_1, M_2 : \tau$.

Type reducibility

Definition: Type τ is *reducible* to type σ iff there exists a closed term $\Phi : \tau \to \sigma$ such that the operator $\lambda\!\!\lambda M{:}\tau.\, \Phi M$ is injective on closed terms, i.e.,

$$\Phi M_1 =_{\beta\eta} \Phi M_2 \quad \text{implies} \quad M_1 =_{\beta\eta} M_2$$

for closed $M_1, M_2 : \tau$.

Theorem (R. Statman):

Every type over a single type constant $\mathbf{0}$ *is reducible to* **tree**.

Semantics for finite types

Assumptions:

- Orthodox Church style;
- Only one atomic type $\mathbf{0}$;
- Extensional equality $=_{\beta\eta}$.

○

Standard model $\mathfrak{M}(A)$

- Basic domain $D_0 = A$;
- Function domains: $D_{\sigma\to\tau} = D_\sigma \to D_\tau$;
- Obvious semantics:
  - $[\![x]\!]_v = v(x)$;
  - $[\![MN]\!]_v = [\![M]\!]_v([\![N]\!]_v)$;
  - $[\![\lambda x^\tau .\, M]\!]_v = \lambda\!\!\lambda d{\in}D_\tau .\, [\![M]\!]_{v[x\mapsto d]}$.

∘

Completeness

Theorem (Harvey Friedman):

Terms are $\beta\eta$-equal iff they are equal in $\mathfrak{M}(\mathbb{N})$.

Completeness

Theorem (Harvey Friedman):

Terms are $\beta\eta$-equal iff they are equal in $\mathfrak{M}(\mathbb{N})$.

Proof:

Define partial surjections $\varphi_\sigma : D_\sigma \multimap T_\sigma/_{=_{\beta\eta}}$ by induction:

For $\sigma = \mathbf{0}$ take $\varphi_{\mathbf{0}} : \mathbb{N} \to T_{\mathbf{0}}/_{=_{\beta\eta}}$ to be any (total) surjection.
(Terms of base type are represented by their numbers.)

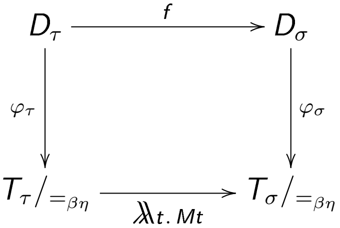
For function types, we represent (the behaviour of) lambda-terms using integer functions, so that:

$$\varphi_\sigma(ab) = \varphi_{\tau\to\sigma}(a)\varphi_\tau(b).$$

○

Completeness proof

Given $\varphi_\sigma : D_\sigma \mathrel{-\!\!\circ\!\!\!\rightarrow} T_\sigma/_{=_{\beta\eta}}$ and $\varphi_\tau : D_\tau \mathrel{-\!\!\circ\!\!\!\rightarrow} T_\tau/_{=_{\beta\eta}}$, we say that a function *$f : D_\tau \rightarrow D_\sigma$ represents a term $M^{\tau\rightarrow\sigma}$* when (informally) the following diagram commutes:



For any M, there exists such an f (not unique).
For a given f, such an M (if exists) is unique up to $\beta\eta$.

Completeness proof

Define partial surjections $\varphi_\sigma : D_\sigma \multimap\!\!\rightarrow T_\sigma/_{=_{\beta\eta}}$ by induction:

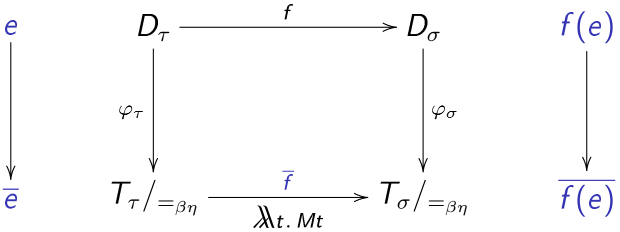
- $\varphi_{\mathbf{0}} : \mathbb{N} \rightarrow T_{\mathbf{0}}/_{=_{\beta\eta}}$ is any (total) surjection.
- $\varphi_{\tau\rightarrow\sigma}(f) = [M]_{=_{\beta\eta}}$ when f represents M.

Abbreviation: If $d \in D_\sigma$, write $\overline{d}$ for $\varphi_\sigma(d)$.

Main property:

If $\overline{f}$ and $\overline{e}$ are defined then $\overline{f(e)}$ is defined and $\overline{f}\,\overline{e} =_{\beta\eta} \overline{f(e)}$

“Partial epimorphism”: $\overline{f}\,\overline{e} =_{\beta\eta} \overline{f(e)}$



Completeness proof

Lemma:
Take v so that $\overline{v(x)} = x$, for all x. Then $M =_{\beta\eta} \overline{[\![M]\!]_v}$, all M.

Completeness proof

Lemma:
Take v so that $\overline{v(x)} = x$, for all x. Then $M =_{\beta\eta} \overline{[\![M]\!]_v}$, all M.

Main Proof: Let $\mathfrak{M}(\mathbb{N}) \models M = N$. Then $[\![M]\!]_v = [\![N]\!]_v$, for all v, in particular for v as above. Therefore

$$M =_{\beta\eta} \overline{[\![M]\!]_v} =_{\beta\eta} \overline{[\![N]\!]_v} =_{\beta\eta} N.$$

○

Finite completeness

Theorem (R. Statman):

For every M there is k such that, for all N:

$$M =_{\beta\eta} N \qquad \textit{iff} \qquad \mathfrak{M}(k) \models M = N.$$

Corollary:

Terms are $\beta\eta$-equal iff they are equal in all finite models.

Finite completeness proof

It suffices to prove that

for every closed M : **tree** *there is* k *such that, for all* N : **tree**:

$$M =_{\beta\eta} N \qquad \textit{iff} \qquad \mathfrak{M}(k) \models M = N.$$

Indeed, for closed $M : \tau$, consider $\Phi(M)$,
where Φ is a reduction of τ to **tree**.

For non-closed terms, consider appropriate lambda-closures.

$\circ$

Let $p(m)(n) = 2^m(2n+1)$. Then $p \in D_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}$ in $\mathfrak{M}(\mathbb{N})$.
Observe that $p(m)(n) > m, n$, for all m, n.

Let $p(m)(n) = 2^m(2n+1)$. Then $p \in D_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}$ in $\mathfrak{M}(\mathbb{N})$.
Observe that $p(m)(n) > m, n$, for all m, n.

Term zamknięty typu **tree**, to w istocie drzewo.

Let $p(m)(n) = 2^m(2n+1)$. Then $p \in D_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}$ in $\mathfrak{M}(\mathbb{N})$.
Observe that $p(m)(n) > m, n$, for all m, n.

Term zamknięty typu **tree**, to w istocie drzewo.

Wartość $[\![M]\!](p)(0)$ można uważać za numer tego drzewa.

Ćwiczenie: Jaka liczba jest numerem drzewa

$$\lambda px.\, px(p(pxx)x)?$$

○

For M : **tree**, define $k = 2 + [\![M]\!](p)(0)$, i.e. $2 + numer(M)$.

For M : **tree**, define $k = 2 + [\![M]\!](p)(0)$, i.e. $2 + \mathit{numer}(M)$.

Let $p' : k \to k \to k$ be p „truncated" to values less than k.
Then $p' \in D_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}$ in $\mathfrak{M}(k)$.

For M : **tree**, define $k = 2 + [\![M]\!](p)(0)$, i.e. $2 + \mathit{numer}(M)$.

Let $p' : k \to k \to k$ be p „truncated" to values less than k.
Then $p' \in D_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}$ in $\mathfrak{M}(k)$.

Suppose $\mathfrak{M}(k) \models M = N$. Then in the model $\mathfrak{M}(k)$:

$$k - 2 = [\![M]\!](p')(0) = [\![N]\!](p')(0) \qquad (*)$$

For M : **tree**, define $k = 2 + [\![M]\!](p)(0)$, i.e. $2 + \mathit{numer}(M)$.

Let $p' : k \to k \to k$ be p „truncated" to values less than k.
Then $p' \in D_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}$ in $\mathfrak{M}(k)$.

Suppose $\mathfrak{M}(k) \models M = N$. Then in the model $\mathfrak{M}(k)$:

$$k - 2 = [\![M]\!](p')(0) = [\![N]\!](p')(0) \qquad (*)$$

But all numbers needed to verify (*) are at most $k - 2$.
(Otherwise the rhs equals $k - 1$.)

For M : **tree**, define $k = 2 + [\![M]\!](p)(0)$, i.e. $2 + \mathit{numer}(M)$.

Let $p' : k \to k \to k$ be p „truncated" to values less than k.
Then $p' \in D_{\mathbf{0}\to\mathbf{0}\to\mathbf{0}}$ in $\mathfrak{M}(k)$.

Suppose $\mathfrak{M}(k) \models M = N$. Then in the model $\mathfrak{M}(k)$:

$$k - 2 = [\![M]\!](p')(0) = [\![N]\!](p')(0) \qquad (*)$$

But all numbers needed to verify (*) are at most $k - 2$.
(Otherwise the rhs equals $k - 1$.)

Therefore $[\![M]\!](p)(0) = [\![N]\!](p)(0)$ holds also in $\mathfrak{M}(\mathbb{N})$.
It follows that $M =_{\beta\eta} N$.

○

Equality is not definable in simple types

There is no $E : \omega_\tau \to \omega_\sigma \to \omega_\rho$*, such that for all* $p, q \in \mathbb{N}$*:*

$$E\,\mathbf{p}^{\omega_\tau}\,\mathbf{q}^{\omega_\sigma} =_{\beta\eta} \mathbf{0}^{\omega_\rho} \qquad \text{iff} \qquad p = q.$$

Equality is not definable in simple types

There is no $E : \omega_\tau \to \omega_\sigma \to \omega_\rho$*, such that for all* $p, q \in \mathbb{N}$*:*

$$E\,\mathbf{p}^{\omega_\tau}\mathbf{q}^{\omega_\sigma} =_{\beta\eta} \mathbf{0}^{\omega_\rho} \qquad \text{iff} \qquad p = q.$$

Proof: By Statman's thm., take k such that for all $N : \omega_\rho$:

$$\mathfrak{M}(k) \models \mathbf{0}^{\omega_\rho} = N \qquad \text{iff} \qquad \mathbf{0}^{\omega_\rho} =_{\beta\eta} N.$$

Equality is not definable in simple types

There is no $E : \omega_\tau \to \omega_\sigma \to \omega_\rho$*, such that for all* $p, q \in \mathbb{N}$*:*

$$E\,\mathbf{p}^{\omega_\tau}\,\mathbf{q}^{\omega_\sigma} =_{\beta\eta} \mathbf{0}^{\omega_\rho} \qquad \text{iff} \qquad p = q.$$

Proof: By Statman's thm., take k such that for all $N : \omega_\rho$:

$$\mathfrak{M}(k) \models \mathbf{0}^{\omega_\rho} = N \qquad \text{iff} \qquad \mathbf{0}^{\omega_\rho} =_{\beta\eta} N.$$

There are $p \neq q$ with $[\![\mathbf{p}^{\omega_\tau}]\!] = [\![\mathbf{q}^{\omega_\tau}]\!]$ in $\mathfrak{M}(k)$.

Equality is not definable in simple types

There is no $E : \omega_\tau \to \omega_\sigma \to \omega_\rho$*, such that for all* $p, q \in \mathbb{N}$*:*

$$E\,\mathbf{p}^{\omega_\tau}\mathbf{q}^{\omega_\sigma} =_{\beta\eta} \mathbf{0}^{\omega_\rho} \qquad \text{iff} \qquad p = q.$$

Proof: By Statman's thm., take k such that for all $N : \omega_\rho$:

$$\mathfrak{M}(k) \models \mathbf{0}^{\omega_\rho} = N \qquad \text{iff} \qquad \mathbf{0}^{\omega_\rho} =_{\beta\eta} N.$$

There are $p \neq q$ with $[\![\mathbf{p}^{\omega_\tau}]\!] = [\![\mathbf{q}^{\omega_\tau}]\!]$ in $\mathfrak{M}(k)$. So in $\mathfrak{M}(k)$:

$$[\![E\mathbf{p}^{\omega_\tau}\mathbf{q}^{\omega_\sigma}]\!] = [\![E]\!][\![\mathbf{p}^{\omega_\tau}]\!][\![\mathbf{q}^{\omega_\sigma}]\!] = [\![E]\!][\![\mathbf{q}^{\omega_\tau}]\!][\![\mathbf{q}^{\omega_\sigma}]\!] = [\![E\mathbf{q}^{\omega_\tau}\mathbf{q}^{\omega_\sigma}]\!] = [\![\mathbf{0}^{\omega_\rho}]\!]$$

Thus $\mathfrak{M}(k) \models E\mathbf{p}^{\omega_\tau}\mathbf{q}^{\omega_\sigma} = \mathbf{0}^{\omega_\rho}$, whence $p = q$.

○

Plotkin's problem

Given $d \in D_\tau$ in a finite model $\mathfrak{M}(X)$.
Is there a term $M : \tau$ with $[\![M]\!] = d$?

More generally:

Let $v(x_1) = e_1 \in D_{\sigma_1}, \ldots, v(x_n) = e_n \in D_{\sigma_n}$.
Is there M such that $[\![M]\!]_v = d$?

(Is d definable from $e_1, \ldots, e_n$?)

Fact: These decision problems are reducible to each other.

Undecidablility of lambda-definability

Theorem (Ralph Loader, 1993):

Plotkin's problem is undecidable.

Undecidablility of lambda-definability

Theorem (Ralph Loader, 1993):

Plotkin's problem is undecidable.

Proof: Reduction from the undecidable word problem for Semi-Thue systems.

Undecidablility of lambda-definability

Theorem (Ralph Loader, 1993):

Plotkin's problem is undecidable.

Proof: Reduction from the undecidable word problem for Semi-Thue systems.

Semi-Thue system: a finite set of *rules* $C \Rightarrow D$, where $C, D \subseteq \{a, b\}^*$. Induces rewriting $xCy \rightarrow xDy$, for any x, y.

Word problem: Can a word w be rewritten to v in a finite number of steps?

○

Undecidablility of lambda-definability

Theorem (Ralph Loader, 1993):

Plotkin's problem is undecidable.

Proof: Reduction from the undecidable word problem for Semi-Thue systems.

Kodujemy słowa w i v i reguły systemu jako elementy modelu
Pytamy, czy v jest definiowalne z w i reguł.

○

Proof

Take $X = \{a, b, L, R, *, 1, 0\}$. Encode any word $w = o_1 \dots o_n$ as a function $\overline{w} : D_0^n \to D_0$, such that

- $\overline{w}(* \cdots * \, o_i * \cdots *) = 1$, if the i-th symbol in w is o_i;
- $\overline{w}(* \cdots * \, LR * \cdots *) = 1$;
- $\overline{w}(\dots) = 0$, otherwise.

○

How does it work?

For $w = w_1 C w_2$ we have $\overline{w} = \lambda\!\!\lambda \vec{x} \lambda\!\!\lambda \vec{y} \lambda\!\!\lambda \vec{z}.\, \overline{w}(\vec{x})(\vec{y})(\vec{z})$.

How does it work?

For $w = w_1 C w_2$ we have $\overline{w} = \lambda\!\!\lambda \vec{x} \lambda\!\!\lambda \vec{y} \lambda\!\!\lambda \vec{z}.\, \overline{w}(\vec{x})(\vec{y})(\vec{z})$.

Fix $\vec{x}, \vec{z}$ and consider the function $g = \lambda\!\!\lambda \vec{y}.\, \overline{w}(\vec{x})(\vec{y})(\vec{z})$.

It "accepts" the following strings (depending on $\vec{x}, \vec{z}$):

$\vec{x}$	$\vec{y}$	$\vec{z}$
$* \cdots * \, o_i \, * \cdots *$	$* \cdots\cdots *$	$* \cdots\cdots *$
$* \cdots\cdots *$	same as $\overline{C}$	$* \cdots\cdots *$
$* \cdots\cdots *$	$* \cdots\cdots *$	$* \cdots * \, o_i \, * \cdots *$
$* \cdots * \, LR \, * \cdots *$	$* \cdots\cdots *$	$* \cdots\cdots *$
$* \cdots\cdots * \, L$	$R \, * \cdots\cdots *$	$* \cdots\cdots *$
$* \cdots\cdots *$	$* \cdots\cdots * \, L$	$R \, * \cdots\cdots *$
$* \cdots\cdots *$	$* \cdots\cdots *$	$* \cdots * \, LR \, * \cdots *$

How does it work?

Fix $\vec{x}, \vec{z}$ and consider the function $g = \lambda\!\!\lambda \vec{y}.\, \overline{w}(\vec{x})(\vec{y})(\vec{z})$.

Depending on $\vec{x}, \vec{z}$, the function g is as follows:

$\vec{x}$	g	$\vec{z}$
$* \cdots * \, o_i \, * \cdots *$	$\chi_{\{*\cdots*\}}$	$* \cdots\cdots *$
$* \cdots\cdots *$	$\overline{C}$	$* \cdots\cdots *$
$* \cdots\cdots *$	$\chi_{\{*\cdots*\}}$	$* \cdots * \, o_i \, * \cdots *$
$* \cdots * \, LR \, * \cdots *$	$\chi_{\{*\cdots*\}}$	$* \cdots\cdots *$
$* \cdots\cdots * \, L$	$\chi_{\{R*\cdots*\}}$	$* \cdots\cdots *$
$* \cdots\cdots *$	$\chi_{\{*\cdots*L\}}$	$R \, * \cdots\cdots *$
$* \cdots\cdots *$	$\chi_{\{*\cdots*\}}$	$* \cdots * \, LR \, * \cdots *$

Otherwise $g = \chi_{\varnothing}$.

How to encode a rule $F = (C \Rightarrow D)$?

How to encode a rule $F = (C \Rightarrow D)$?

Fix $\vec{x}, \vec{z}$ and consider the function $g = \lambda\!\!\lambda \vec{y}.\, \overline{w}(\vec{x})(\vec{y})(\vec{z})$.

What will change in this table if we replace $w_1 C w_2$ by $w_1 D w_2$?

$\vec{x}$	g	$\vec{z}$
$* \cdots * \, o_i \, * \cdots *$	$\chi_{\{* \cdots *\}}$	$* \cdots\cdots *$
$* \cdots\cdots *$	$\overline{C}$	$* \cdots\cdots *$
$* \cdots\cdots *$	$\chi_{\{* \cdots *\}}$	$* \cdots * \, o_i \, * \cdots *$
$* \cdots * \, LR \, * \cdots *$	$\chi_{\{* \cdots *\}}$	$* \cdots\cdots *$
$* \cdots\cdots * \, L$	$\chi_{\{R * \cdots *\}}$	$* \cdots\cdots *$
$* \cdots\cdots *$	$\chi_{\{* \cdots * L\}}$	$R \, * \cdots\cdots *$
$* \cdots\cdots *$	$\chi_{\{* \cdots *\}}$	$* \cdots * \, LR \, * \cdots *$
otherwise	$\chi_{\varnothing}$	otherwise

How to encode a rule $F = (C \Rightarrow D)$

Every rule $F = (C \Rightarrow D)$ is encoded as a function

$$\overline{F} : (D_0^m \to D_0) \to (D_0^n \to D_0),$$

where $m = |C|$ and $n = |D|$. We take:

- $\overline{F}(\chi_{\{*\cdots*\}}) = \chi_{\{*\cdots*\}}$;
- $\overline{F}(\chi_{\{R*\cdots*\}}) = \chi_{\{R*\cdots*\}}$;
- $\overline{F}(\chi_{\{*\cdots*L\}}) = \chi_{\{*\cdots*L\}}$;
- $\overline{F}(\overline{C}) = \overline{D}$;
- $\overline{F}(g) = \chi_{\varnothing}$, for any other g.

Claim

A word w can be rewritten to v iff the element $\overline{v}$ of $\mathfrak{M}(X)$ is definable from $\overline{w}$ and the functions $\overline{F}$ encoding the rules.

Claim

A word w can be rewritten to v iff the element $\overline{v}$ of $\mathfrak{M}(X)$ is definable from $\overline{w}$ and the functions $\overline{F}$ encoding the rules.

The easy part: Let $w = w_1 C w_2$ rewrites to $v = w_1 D w_2$ using $F = (C \Rightarrow D)$. Assume that term W defines $\overline{w}$. Then $\overline{v}$ is definable by

$$V = \lambda \vec{x}\vec{u}\vec{z}.\, \overline{F}(\lambda \vec{y}.\, W \vec{x}\vec{y}\vec{z})\vec{u}, \qquad (*)$$

It follows that codes of reachable words are definable.

Claim

A word w can be rewritten to v iff the element $\overline{v}$ of $\mathfrak{M}(X)$ is definable from $\overline{w}$ and the functions $\overline{F}$ encoding the rules.

The easy part: Let $w = w_1 C w_2$ rewrites to $v = w_1 D w_2$ using $F = (C \Rightarrow D)$. Assume that term W defines $\overline{w}$. Then $\overline{v}$ is definable by

$$V = \lambda \vec{x}\vec{u}\vec{z}.\ \overline{F}(\lambda \vec{y}.\, W\vec{x}\vec{y}\vec{z})\vec{u}, \qquad (*)$$

It follows that codes of reachable words are definable.

The hard part: And conversely.

○