Instytut Informatyki

Programowanie funkcyjne

Wykład 12. Rachunek λ z typami prostymi. Izomorfizm Curry'ego-Howarda.

Zdzisław Spławski

- Dowody konstruktywne
- Dedukcja naturalna dla rachunku zdań
- Rachunek lambda z typami prostymi $\lambda_{\rightarrow}$
- Izomorfizm Curry'ego-Howarda
- Rachunek lambda z typami prostymi $\lambda_{\rightarrow\wedge\vee\perp\neg}$
- Przykłady dla $\lambda_{\rightarrow\wedge\vee\perp\neg}$
- Własności rachunku lambda z typami prostymi
- Izomorfizm Curry'ego-Howarda w języku OCaml
- Silna typizacja
- Let-polimorfizm
- Literatura

Twierdzenie. Istnieją dwie niewymierne liczby a i b takie, że a^b jest wymierne.
Dowód. Albo $(\sqrt{2})^{\sqrt{2}}$ jest wymierne i wtedy $a = b = \sqrt{2}$ albo $(\sqrt{2})^{\sqrt{2}}$ jest niewymierne i wtedy $a = (\sqrt{2})^{\sqrt{2}}, b = \sqrt{2}$. ■

Dowód tego twierdzenie jest oczywiście poprawny w logice klasycznej. Jest to przykład dowodu niekonstruktywnego, ze względu na wykorzystanie reguły wykluczonego środka. Pomimo, że twierdzenie jest dowiedzione i wiadomo, że liczby a i b istnieją, to nie można (na podstawie tego dowodu) znaleźć pary liczb spełniających warunek określony w twierdzeniu.
Konstruktywny dowód tego twierdzenia jest oparty na twierdzeniu Gelfonda–Schneidera.Wynika z niego, że liczby $a = (\sqrt{2})^{\sqrt{2}}, b = \sqrt{2}$ spełniają powyższe twierdzenie.
Dowody konstruktywne nie tylko dowodzą istnienia obiektów matematycznych, lecz także zawierają metodę ich skonstruowania.
Najbardziej znaną *logiką konstruktywną* jest *logika intuicjonistyczna*, wprowadzona przez Brouwera i rozwijana przez Heytinga i wielu innych. Znajduje ona wiele zastosowań w informatyce (*izomorfizm Curry'ego-Howarda*).

Reguły wnioskowania w notacji sekwentowej I

$$\frac{}{\varphi \vdash \varphi}\,(Ass)$$

$$\frac{\Gamma \vdash \varphi}{\Gamma, \psi \vdash \varphi}\,(W)$$

reguły wprowadzania	reguły eliminacji
$\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\,(\to I)$	$\dfrac{\Gamma_1 \vdash \varphi \to \psi \qquad \Gamma_2 \vdash \varphi}{\Gamma_1, \Gamma_2 \vdash \psi}\,(\to E)$
$\dfrac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}\,(\neg I)$	$\dfrac{\Gamma_1 \vdash \neg\varphi \qquad \Gamma_2 \vdash \varphi}{\Gamma_1, \Gamma_2 \vdash \bot}\,(\neg E)$

Reguły wnioskowania w notacji sekwentowej II

reguły wprowadzania	reguły eliminacji
$\dfrac{\Gamma_1 \vdash \varphi \qquad \Gamma_2 \vdash \psi}{\Gamma_1, \Gamma_2 \vdash \varphi \land \psi}\ (\land I)$	$\dfrac{\Gamma \vdash \varphi \land \psi}{\Gamma \vdash \varphi}\ (\land E_1)$
	$\dfrac{\Gamma \vdash \varphi \land \psi}{\Gamma \vdash \psi}\ (\land E_2)$
$\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \lor \psi}\ (\lor I_1)$	
$\dfrac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \lor \psi}\ (\lor I_2)$	$\dfrac{\Gamma_1 \vdash \varphi \lor \psi \qquad \Gamma_2, \varphi \vdash \rho \qquad \Gamma_3, \psi \vdash \rho}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \rho}\ (\lor E)$
$\dfrac{}{\vdash \top}\ (\top I)$	brak
brak	$\dfrac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\ (\bot E)$

Reguły wnioskowania w notacji sekwentowej III

reguły wprowadzania	reguły eliminacji
$\dfrac{\Gamma_1, \varphi \vdash \psi \qquad \Gamma_2, \psi \vdash \varphi}{\Gamma_1, \Gamma_2 \vdash \varphi \leftrightarrow \psi}\ (\leftrightarrow I)$	$\dfrac{\Gamma_1 \vdash \varphi \leftrightarrow \psi \qquad \Gamma_2 \vdash \varphi}{\Gamma_1, \Gamma_2 \vdash \psi}\ (\leftrightarrow E_1)$
	$\dfrac{\Gamma_1 \vdash \varphi \leftrightarrow \psi \qquad \Gamma_2 \vdash \psi}{\Gamma_1, \Gamma_2 \vdash \varphi}\ (\leftrightarrow E_2)$

dla logiki klasycznej
$\dfrac{\Gamma, \neg\varphi \vdash \bot}{\Gamma \vdash \varphi}\ (RAA)$ lub $\dfrac{\Gamma \vdash \neg\neg\varphi}{\Gamma \vdash \varphi}\ (\neg\neg)$ lub $\dfrac{}{\vdash \varphi \vee \neg\varphi}\ (TND)$

(RAA) — reductio ad absurdum (reguła sprowadzenia do sprzeczności)

(TND) — tertium non datur (reguła wykluczonego środka)

$(\neg\neg)$ — reguła podwójnego zaprzeczenia

Reguły wnioskowania w notacji założeniowej I

reguły wprowadzania	reguły eliminacji
$\dfrac{\varphi \qquad \psi}{\varphi \wedge \psi}\ (\wedge I)$	$\dfrac{\varphi \wedge \psi}{\varphi}\ (\wedge E_1) \qquad \dfrac{\varphi \wedge \psi}{\psi}\ (\wedge E_2)$
$\dfrac{\varphi}{\varphi \vee \psi}\ (\vee I_1) \qquad \dfrac{\psi}{\varphi \vee \psi}\ (\vee I_2)$	$\dfrac{\varphi \vee \psi \qquad \begin{matrix}[\varphi]^n \\ \vdots \\ \rho\end{matrix} \qquad \begin{matrix}[\psi]^n \\ \vdots \\ \rho\end{matrix}}{\rho}\ {}_n(\vee E)$
$\dfrac{}{\top}\ (\top I)$	brak
brak	$\dfrac{\perp}{\varphi}\ (\perp E)$

Reguły wnioskowania w notacji założeniowej II

reguły wprowadzania	reguły eliminacji
$\dfrac{\begin{array}{c}[\varphi]^n \\ \vdots \\ \psi\end{array}}{\varphi \to \psi}\ {}_n(\to I)$	$\dfrac{\varphi \to \psi \qquad \varphi}{\psi}\ (\to E)$
$\dfrac{\begin{array}{c}[\varphi]^n \\ \vdots \\ \bot\end{array}}{\neg\varphi}\ {}_n(\neg I)$	$\dfrac{\neg\varphi \qquad \varphi}{\bot}\ (\neg E)$

Reguły wnioskowania w notacji założeniowej III

reguły wprowadzania	reguły eliminacji
$\dfrac{\begin{matrix}[\varphi]^n & [\psi]^n \\ \vdots & \vdots \\ \psi & \varphi\end{matrix}}{\varphi \leftrightarrow \psi}\ {}_n(\leftrightarrow I)$	$\dfrac{\varphi \leftrightarrow \psi \qquad \varphi}{\psi}\ (\leftrightarrow E_1) \qquad \dfrac{\varphi \leftrightarrow \psi \qquad \psi}{\varphi}\ (\leftrightarrow E_2)$

dla logiki klasycznej

$$\dfrac{\begin{matrix}[\neg\varphi]^n \\ \vdots \\ \bot\end{matrix}}{\varphi}\ {}_n(RAA) \qquad \text{lub} \qquad \dfrac{\neg\neg\varphi}{\varphi}\ (\neg\neg) \qquad \text{lub} \qquad \dfrac{}{\varphi \vee \neg\varphi}\ (TND)$$

(RAA) — reductio ad absurdum (reguła sprowadzenia do sprzeczności)
(TND) — tertium non datur (reguła wykluczonego środka)

$(\neg\neg)$ — reguła podwójnego zaprzeczenia

Prawo symplifikacji: $\vdash \varphi \to \psi \to \varphi$

Notacja sekwentowa.

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{p \vdash p}\,(Ass)}{p, q \vdash p}\,(W)}{p \vdash q \to p}\,(\to I)}{\vdash p \to q \to p}\,(\to I)
$$

Prawo symplifikacji: $\vdash \varphi \to \psi \to \varphi$

Notacja założeniowa.

$$\dfrac{\dfrac{[p]^1}{q \to p}\,(\to I)}{p \to q \to p}\,{}_1(\to I)$$

Uwaga. W notacji założeniowej możliwe jest użycie reguły $(\to I)$ bez zamykania żadnego założenia.

Prawo sylogizmu hipotetycznego (przechodniość implikacji):

$\vdash (\varphi \to \psi) \to (\psi \to \rho) \to \varphi \to \rho$

Notacja sekwentowa.

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(Ass)}
\UnaryInfC{$\psi \to \rho \vdash \psi \to \rho$}
\AxiomC{}
\RightLabel{(Ass)}
\UnaryInfC{$\varphi \to \psi \vdash \varphi \to \psi$}
\AxiomC{}
\RightLabel{(Ass)}
\UnaryInfC{$\varphi \vdash \varphi$}
\RightLabel{$(\to E)$}
\BinaryInfC{$\varphi \to \psi, \varphi \vdash \psi$}
\RightLabel{$(\to E)$}
\BinaryInfC{$\psi \to \rho, \varphi \to \psi, \varphi \vdash \rho$}
\RightLabel{$(\to I)$}
\UnaryInfC{$\psi \to \rho, \varphi \to \psi \vdash \varphi \to \rho$}
\RightLabel{$(\to I)$}
\UnaryInfC{$\varphi \to \psi \vdash (\psi \to \rho) \to \varphi \to \rho$}
\RightLabel{$(\to I)$}
\UnaryInfC{$\vdash (\varphi \to \psi) \to (\psi \to \rho) \to \varphi \to \rho$}
\end{prooftree}
$$

Prawo sylogizmu hipotetycznego (przechodniość implikacji):

$\vdash (\varphi \to \psi) \to (\psi \to \rho) \to \varphi \to \rho$

Notacja założeniowa.

$$
\dfrac{\dfrac{\dfrac{\dfrac{[\psi \to \rho]^1 \qquad \dfrac{[\varphi \to \psi]^2 \qquad [\varphi]^3}{\psi}\,(\to E)}{\rho}\,(\to E)}{\varphi \to \rho}\,{}_3(\to I)}{(\psi \to \rho) \to \varphi \to \rho}\,{}_1(\to I)}{(\varphi \to \psi) \to (\psi \to \rho) \to \varphi \to \rho}\,{}_2(\to I)
$$

Prawo importacji: $\vdash (\varphi \to \psi \to \rho) \to (\varphi \land \psi \to \rho)$

Notacja sekwentowa.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p \to q \to r \vdash p \to q \to r}\,(Ass) \qquad \dfrac{\dfrac{}{p \land q \vdash p \land q}\,(Ass)}{p \land q \vdash p}\,(\land E)}{p \to q \to r, p \land q \vdash q \to r}\,(\to E) \qquad \dfrac{\dfrac{}{p \land q \vdash p \land q}\,(Ass)}{p \land q \vdash q}\,(\land E)}{p \to q \to r, p \land q \vdash r}\,(\to E)}{p \to q \to r \vdash p \land q \to r}\,(\to I)}{\vdash (p \to q \to r) \to (p \land q \to r)}\,(\to I)$$

Prawo importacji: $\vdash (\varphi \to \psi \to \rho) \to (\varphi \wedge \psi \to \rho)$

Notacja założeniowa.

$$
\dfrac{\dfrac{\dfrac{\dfrac{[\varphi \to \psi \to \rho]^2 \qquad \dfrac{[\varphi \wedge \psi]^1}{\varphi}\,(\wedge E)}{\psi \to \rho}\,(\to E) \qquad \dfrac{[\varphi \wedge \psi]^1}{\psi}\,(\wedge E)}{\rho}\,(\to E)}{\varphi \wedge \psi \to \rho}\,{}_1(\to I)}{(\varphi \to \psi \to \rho) \to (\varphi \wedge \psi \to \rho)}\,{}_2(\to I)
$$

Definicje I

Typy proste są budowane z typów atomowych $\mathbb{A}$ za pomocą konstruktora przestrzeni funkcyjnej $\to$. Zbiór typów atomowych w systemach teoretycznych to najczęściej nieskończony zbiór zmiennych dla typów $\mathbb{A} = \{\alpha_0, \alpha_1, \alpha_2, \ldots\}$ lub zbiór jednoelementowy $\mathbb{A} = \{0\}$. W systemach stosowanych wykorzystywane są też typy Bool, Nat, Real itp.

Asercja typizująca (ang. typing assertion) ma postać:

$$\Gamma \vdash M : \tau$$

gdzie Γ oznacza *kontekst typizujący* (ang. typing context)

$$\Gamma \equiv \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$$

w którym każda zmienna występuje co najwyżej raz. Kontekst typizujący zawiera (co najmniej) informację o typach wszystkich zmiennych wolnych termu M.

$\Gamma \vdash M : \tau$ można czytać "w kontekście Γ term M ma typ τ".

Definicje II

Istnieją dwa podejścia do rachunku lambda z typami.

- *Typizacja w stylu Curry'ego* lub *typizacja niejawna* (ang. Curry style typing, implicit typing); termy są takie same jak w beztypowym rachunku lambda, a ich typy są konstruowane za pomocą odpowiedniego algorytmu.
- *Typizacja w stylu Churcha* lub *typizacja jawna* (ang. Church style typing, explicit typing); informacja o typach (w szczególności o typach zmiennych związanych) jest jawnie podawana w termach.

W dalszej części wykładu będzie stosowane podejście Curry'ego.

Definicje III

Definicja

Niech $\mathbb{A}$ będzie niepustym zbiorem typów atomowych. Zbiór typów prostych $\mathbb{T} = \mathbb{T}^{\mathbb{A}}$ nad $\mathbb{A}$ jest definiowany indukcyjnie następująco:

$$
\begin{array}{rcll}
\alpha \in \mathbb{A} & \Rightarrow & \alpha \in \mathbb{T} & \quad \textit{typy atomowe} \\
\sigma, \tau \in \mathbb{T} & \Rightarrow & (\sigma \to \tau) \in \mathbb{T} & \quad \textit{typy funkcyjne}
\end{array}
$$

Reguły typizacji. Notacja sekwentowa.

$$\frac{}{x : \tau \vdash x : \tau}\ (\mathrm{Ass}) \qquad \frac{\Gamma \vdash M : \tau}{\Gamma, x : \sigma \vdash M : \tau}\ (\mathrm{W})$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to I)$$

$$\frac{\Gamma_1 \vdash M : \sigma \to \tau \qquad \Gamma_2 \vdash N : \sigma}{\Gamma_1, \Gamma_2 \vdash MN : \tau}\ (\to E)$$

Reguły typizacji. Notacja założeniowa.

reguła wprowadzania	reguła eliminacji
$$\dfrac{\begin{array}{c}[x:\varphi]\\ \vdots \\ M:\psi\end{array}}{\lambda x.M:\varphi\to\psi}\;(\to I)$$	$$\dfrac{M:\varphi\to\psi \qquad N:\varphi}{MN:\psi}\;(\to E)$$

Izomorfizm Curry'ego-Howarda

Istnieje ścisła odpowiedniość między formułami logiki konstruktywnej, a typami w rachunku lambda z typami, która jest użyteczna zarówno w teorii dowodu (ang. proof theory), jak i w programowaniu. W logice intuicjonistycznej znaczenie (semantyka) formuły jest definiowana za pomocą (konstruktywnego) dowodu (co trzeba zrobić, żeby udowodnić tę formułę) i jest znana jako interpretacja BHK (Brouwer, Heyting, Kołmogorow). Niezmiennikiem reguł wnioskowania jest więc *konstruowalność*, w odróżnieniu od logiki klasycznej, gdzie niezmiennikiem jest *prawdziwość* formuły. Związki takiej interpretacji z programowaniem pokazują poniższe odpowiedniości:

$$\begin{array}{rcccl} \text{formuła} & = & \text{typ} & = & \text{specyfikacja} \\ \text{dowód} & = & \text{term} & = & \text{program} \end{array}$$

Ta odpowiedniość jest nazywana "izomorfizmem Curry'ego-Howarda" (ang. "Curry-Howard isomorphism", Curry-Howard correspondence, propositions-as-types correspondence).

Prawo symplifikacji: $\vdash \varphi \to \psi \to \varphi$

Notacja sekwentowa.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{x : p \vdash x : p}\ (Ass)}{x : p, y : q \vdash x : p}\ (W)}{x : p \vdash \lambda y.x : q \to p}\ (\to I)}{\vdash \lambda x.\lambda y.x : p \to q \to p}\ (\to I)$$

Prawo symplifikacji = kombinator K (wykład 10, str. 12)

Prawo sylogizmu hipotetycznego (przechodniość implikacji):

$\vdash (\varphi \to \psi) \to (\psi \to \rho) \to \varphi \to \rho$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f : \psi \to \rho \vdash f : \psi \to \rho}\,(Ass) \qquad \dfrac{\dfrac{}{g : \varphi \to \psi \vdash g : \varphi \to \psi}\,(Ass) \qquad \dfrac{}{x : \varphi \vdash x : \varphi}\,(Ass)}{g : \varphi \to \psi, x : \varphi \vdash gx : \psi}\,(\to E)}{f : \psi \to \rho, g : \varphi \to \psi, x : \varphi \vdash f(gx) : \rho}\,(\to E)}{f : \psi \to \rho, g : \varphi \to \psi \vdash \lambda x.f(gx) : \varphi \to \rho}\,(\to I)}{g : \varphi \to \psi \vdash \lambda f x.f(gx) : (\psi \to \rho) \to \varphi \to \rho}\,(\to I)}{\vdash \lambda g f x.f(gx) : (\varphi \to \psi) \to (\psi \to \rho) \to \varphi \to \rho}\,(\to I)
$$

Prawo sylogizmu hipotetycznego = składanie funkcji
Niech będą dane dwie funkcje $f : \psi \to \rho$ oraz $g : \varphi \to \psi$. Złożenie $(f \circ g) : \varphi \to \rho$ jest definiowane następująco:

$$(f \circ g)(x) = f(gx)$$

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{f:\varphi\to\varphi \vdash f:\varphi\to\varphi}\,(Ass)
}{f:\varphi\to\varphi, x:\psi \vdash f:\varphi\to\varphi}\,(W)
}{f:\varphi\to\varphi \vdash \lambda x.f:\psi\to\varphi\to\varphi}\,(\to I)
}{\vdash \lambda fx.f:(\varphi\to\varphi)\to\psi\to\varphi\to\varphi}\,(\to I)
\qquad
\dfrac{
\dfrac{}{y:\varphi\vdash y:\varphi}\,(\mathrm{Ass})
}{\vdash \lambda y.y:\varphi\to\varphi}\,(\to I)
}{\vdash (\lambda fx.f)(\lambda y.y):\psi\to\varphi\to\varphi}\,(\to E)
$$

Przedstawione drzewo wywodu nie jest najprostsze. Wykorzystano założenie $f:\varphi\to\varphi$. Jednak w prawym poddrzewie rzeczywiście *udowodniliśmy* $\varphi\to\varphi$, wobec tego *założenie* $f:\varphi\to\varphi$ właściwie nie było potrzebne. W efekcie otrzymany λ-term zawiera β-redeks. Niżej przedstawiono najkrótszy dowód $\psi\to\varphi\to\varphi$.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{y:\varphi\vdash y:\varphi}\,(Ass)
}{\vdash \lambda y.y:\varphi\to\varphi}\,(\to I)
}{x:\psi\vdash \lambda y.y:\varphi\to\varphi}\,(W)
}{\vdash \lambda xy.y:\psi\to\varphi\to\varphi}\,(\to I)
$$

Normalizacja (upraszczanie) dowodów ≈ redukcja termów

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x:\sigma \vdash x:\sigma}\,(Ass)}{\begin{array}{c}\mathcal{D}_1\\ \Delta, x:\sigma \vdash M:\tau\end{array}}}{\Delta \vdash \lambda x.M : \sigma \to \tau}\,(\to I) \qquad \begin{array}{c}\mathcal{D}_2\\ \Gamma \vdash N:\sigma\end{array}}{\Gamma, \Delta \vdash (\lambda x.M)N : \tau}\,(\to E) \quad \to_\beta \quad \begin{array}{c}\mathcal{D}_2\\ \Gamma \vdash N:\sigma\\ \mathcal{D}_1[x:=N]\\ \Gamma, \Delta \vdash M[x:=N]:\tau\end{array}$$

Bezpośrednio po regule wprowadzania $(\to I)$
została użyta reguła eliminacji $(\to E)$.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash M : \sigma \to \tau\end{array} \qquad \dfrac{}{x:\sigma \vdash x:\sigma}\,(Ass)}{\Gamma, x:\sigma \vdash Mx:\tau}\,(\to E)}{\Gamma \vdash \lambda x.Mx : \sigma \to \tau}\,(\to I) \quad \to_\eta \quad \begin{array}{c}\mathcal{D}\\ \Gamma \vdash M : \sigma \to \tau\end{array}$$

Bezpośrednio po regule eliminacji $(\to E)$
została użyta reguła wprowadzania $(\to I)$.
Zakładamy, że $x \notin FV(M)$.

Izomorfizm Curry'ego-Howarda

Przykład (bardziej skomplikowany)

$$\mathcal{D} = \dfrac{[y:\sigma\to\tau] \qquad [z:\sigma]}{yz:\tau}\,(\to E)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[v:\tau\to\xi] \qquad [u:\tau]}{vu:\xi}\,(\to E)}{\lambda u.vu:\tau\to\xi}\,(\to I)}{\lambda vu.vu:(\tau\to\xi)\to\tau\to\xi}\,(\to I) \qquad \dfrac{[x:\sigma\to\tau\to\xi] \qquad [z:\sigma]}{xz:\tau\to\xi}\,(\to E)}{(\lambda vu.vu)(xz):\tau\to\xi}\,(\to E) \qquad \mathcal{D}}{\dfrac{\dfrac{(\lambda vu.vu)(xz)(yz):\xi}{\lambda z.(\lambda vu.vu)(xz)(yz):\sigma\to\xi}\,(\to I)}{\dfrac{\lambda yz.(\lambda vu.vu)(xz)(yz):(\sigma\to\tau)\to\sigma\to\xi}{\lambda xyz.(\lambda vu.vu)(xz)(yz):(\sigma\to\tau\to\xi)\to(\sigma\to\tau)\to\sigma\to\xi}}\,(\to I)}$$

W otrzymanym λ-termie są zawarte dwa redeksy: η-redeks $\lambda vu.vu$ oraz β-redeks $(\lambda v.M)(xz)$. Po redukcji otrzymamy term $\lambda xyz.xz(yz)$.

Przykładowy wywód po zastosowaniu $\to_\eta$

$$\mathcal{D} = \dfrac{[y : \sigma \to \tau] \qquad\qquad [z : \sigma]}{yz : \tau}\,(\to E)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[v : \tau \to \xi]}{\lambda v.v : (\tau \to \xi) \to \tau \to \xi}\,(\to I) \qquad \dfrac{[x : \sigma \to \tau \to \xi] \qquad [z : \sigma]}{xz : \tau \to \xi}\,(\to E)}{(\lambda v.v)(xz) : \tau \to \xi}\,(\to E) \qquad \mathcal{D}}{(\lambda v.v)(xz)(yz) : \xi}}{\lambda z.(\lambda v.v)(xz)(yz) : \sigma \to \xi}\,(\to I)}{\lambda yz.(\lambda v.v)(xz)(yz) : (\sigma \to \tau) \to \sigma \to \xi}\,(\to I)}{\lambda xyz.(\lambda v.v)(xz)(yz) : (\sigma \to \tau \to \xi) \to (\sigma \to \tau) \to \sigma \to \xi}$$

Przykładowy wywód po zastosowaniu $\to_\beta$

$$\mathcal{D} = \dfrac{[y : \sigma \to \tau] \qquad [z : \sigma]}{yz : \tau}\,(\to E)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : \sigma \to \tau \to \xi] \qquad [z : \sigma]}{xz : \tau \to \xi}\,(\to E) \qquad \mathcal{D}}{xz(yz) : \xi}\,(\to E)}{\lambda z.xz(yz) : \sigma \to \xi}\,(\to I)}{\lambda yz.xz(yz) : (\sigma \to \tau) \to \sigma \to \xi}\,(\to I)}{\lambda xyz.xz(yz) : (\sigma \to \tau \to \xi) \to (\sigma \to \tau) \to \sigma \to \xi}\,(\to I)$$

Glosariusz izomorfizmu Curry'ego-Howarda

logika	rachunek lambda
formuła (formula)	typ (type)
dowód (proof)	term (term)
zmienna zdaniowa	zmienna dla typu
spójnik zdaniowy	konstruktor typu
implikacja (implication)	przestrzeń funkcyjna (function space)
koniunkcja (conjunction)	iloczyn kartezjański (product)
alternatywa (disjunction)	suma rozłączna (disjoint sum)
absurd (absurdity)	typ pusty (empty type)
założenie (assumption)	zmienna obiektowa (object variable)
reguła wprowadzania	konstruktor wartości
reguła eliminacji	destruktor
obejście w dowodzie (proof detour)	redeks (redex)
normalizacja (normalization)	redukcja (reduction)
dowód normalny (normal proof)	postać normalna (normal form)
dowodliwość (provability)	niepustość typu (inhabitation)

Stałe i reguły redukcji dla $\lambda_{\to\wedge\vee\bot\neg}$

- **pair** : $p \to q \to p \wedge q$
- **fst** : $p \wedge q \to p$
- **snd** : $p \wedge q \to q$
- **inl** : $p \to p \vee q$
- **inr** : $q \to p \vee q$
- **case** : $p \vee q \to (p \to r) \to (q \to r) \to r$
- **absurdE** : $\bot \to p$
- **neg** : $(p \to \bot) \to \neg p$
- **negE** : $\neg p \to p \to \bot$

- $(\lambda x.M)N \to M[x := N]$
- **fst**(**pair** $MN) \to M$
- **snd**(**pair** $MN) \to N$
- **case**(**inl** $M)PQ \to PM$
- **case**(**inr** $M)PQ \to QM$
- **negE**(**neg** $M)N \to MN$

Reguły typizacji. Notacja sekwentowa

$$\frac{}{x : \tau \vdash x : \tau}\ (\mathrm{Ass}) \qquad \frac{\Gamma \vdash M : \tau}{\Gamma, x : \sigma \vdash M : \tau}\ (\mathrm{W})$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ (\to I)$$

$$\frac{\Gamma_1 \vdash M : \sigma \to \tau \qquad \Gamma_2 \vdash N : \sigma}{\Gamma_1, \Gamma_2 \vdash MN : \tau}\ (\to E)$$

$$\frac{\Gamma \vdash M : \varphi \qquad \Delta \vdash N : \psi}{\Gamma, \Delta \vdash \mathsf{\textbf{pair}}\ MN : \varphi \wedge \psi}\ (\wedge I)$$

$$\frac{\Gamma \vdash M : \varphi \wedge \psi}{\Gamma \vdash \mathsf{\textbf{fst}}\ M : \varphi}\ (\wedge E_1) \qquad \frac{\Gamma \vdash M : \varphi \wedge \psi}{\Gamma \vdash \mathsf{\textbf{snd}}\ M : \psi}\ (\wedge E_2)$$

Reguły typizacji. Notacja sekwentowa

$$\frac{\Gamma \vdash M : \varphi}{\Gamma \vdash \textbf{inl}\ M : \varphi \vee \psi}\ (\vee I_1) \qquad \frac{\Gamma \vdash M : \psi}{\Gamma \vdash \textbf{inr}\ M : \varphi \vee \psi}\ (\vee I_2)$$

$$\frac{\Gamma \vdash M : \varphi \vee \psi \qquad \Delta_1, x : \varphi \vdash P : \rho \qquad \Delta_2, x : \psi \vdash Q : \rho}{\Gamma, \Delta_1, \Delta_2 \vdash \textbf{case}\ M(\lambda x.P)(\lambda x.Q) : \rho}\ (\vee E)$$

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \textbf{absurdE}\ M : \tau}\ (\bot E)$$

$$\frac{\Gamma, x : \sigma \vdash M : \bot}{\Gamma \vdash \textbf{neg}\ \lambda x.M : \neg\sigma}\ (\neg I)$$

$$\frac{\Gamma_1 \vdash M : \neg\sigma \qquad \Gamma_2 \vdash N : \sigma}{\Gamma_1, \Gamma_2 \vdash \textbf{negE}\ MN : \bot}\ (\neg E)$$

Prawo importacji: $\vdash (\varphi \to \psi \to \rho) \to (\varphi \wedge \psi \to \rho)$

Notacja sekwentowa.

$$\mathcal{D} = \dfrac{\dfrac{}{x : p \wedge q \vdash x : p \wedge q}\,(Ass)}{x : p \wedge q \vdash \textsf{\textbf{snd}}\ x : q}\,(\wedge E)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : p \to q \to r \vdash f : p \to q \to r}\,(Ass) \qquad \dfrac{\dfrac{}{x : p \wedge q \vdash x : p \wedge q}\,(Ass)}{x : p \wedge q \vdash \textsf{\textbf{fst}}\ x : p}\,(\wedge E)}{f : p \to q \to r, x : p \wedge q \vdash f(\textsf{\textbf{fst}}\ x) : q \to r}\,(\to E) \qquad \mathcal{D}}{f : p \to q \to r, x : p \wedge q \vdash f(\textsf{\textbf{fst}}\ x)(\textsf{\textbf{snd}}\ x) : r}}{f : p \to q \to r \vdash \lambda x.f(\textsf{\textbf{fst}}\ x)(\textsf{\textbf{snd}}\ x) : p \wedge q \to r}\,(\to I)$$

$$\dfrac{f : p \to q \to r \vdash \lambda x.f(\textsf{\textbf{fst}}\ x)(\textsf{\textbf{snd}}\ x) : p \wedge q \to r}{\vdash \lambda f.\lambda x.f(\textsf{\textbf{fst}}\ x)(\textsf{\textbf{snd}}\ x) : (p \to q \to r) \to (p \wedge q \to r)}\,(\to I)$$

Prawo importacji = zwijanie funkcji (funkcjonał `uncurry`)

Prawo importacji: $\vdash (\varphi \to \psi \to \rho) \to \varphi \wedge \psi \to \rho$

Notacja założeniowa.

$$\dfrac{\dfrac{\dfrac{[f : \varphi \to \psi \to \rho] \qquad \dfrac{[x : \varphi \wedge \psi]}{\mathsf{fst}\ x : \varphi}\,(\wedge E)}{f(\mathsf{fst}\ x) : \psi \to \rho}\,(\to E) \qquad \dfrac{[x : \varphi \wedge \psi]}{\mathsf{snd}\ x : \psi}\,(\wedge E)}{f(\mathsf{fst}\ x)(\mathsf{snd}\ x) : \rho}\,(\to E)}{\dfrac{\lambda x.f(\mathsf{fst}\ x)(\mathsf{snd}\ x) : \varphi \wedge \psi \to \rho}{\lambda f \lambda x.f(\mathsf{fst}\ x)(\mathsf{snd}\ x) : (\varphi \to \psi \to \rho) \to \varphi \wedge \psi \to \rho}\,(\to I)}\,(\to I)$$

Prawo importacji = zwijanie funkcji (funkcjonał `uncurry`)

Prawo eksportacji: $\vdash (\varphi \wedge \psi \to \rho) \to \varphi \to \psi \to \rho$

$$
\dfrac{\dfrac{\dfrac{\dfrac{[f : \varphi \wedge \psi \to \rho] \qquad \dfrac{[x : \varphi] \qquad [y : \psi]}{\mathsf{pair}\, x\, y : \varphi \wedge \psi}\,(\wedge I)}{f\,(\mathsf{pair}\; x\; y) : \rho}\,(\to E)}{\lambda y. f\,(\mathsf{pair}\; x\; y) : \psi \to \rho}\,(\to I)}{\lambda x y. f\,(\mathsf{pair}\; x\; y) : \varphi \to \psi \to \rho}\,(\to I)}{\lambda f x y. f\,(\mathsf{pair}\; x\; y) : (\varphi \wedge \psi \to \rho) \to \varphi \to \psi \to \rho}\,(\to I)
$$

Prawo eksportacji = rozwijanie funkcji (funkcjonał `curry`)

- Podstawianie. Jeśli $\Gamma, x : \sigma \vdash M : \tau$ oraz $\Gamma \vdash N : \sigma$, to $\Gamma \vdash M[x := N] : \tau$.
- Poprawność redukcji. Jeśli $\Gamma \vdash M : \tau$ oraz $M \twoheadrightarrow_{\beta\eta} N$, to $\Gamma \vdash N : \tau$.
- Silna normalizacja. Jeśli $\Gamma \vdash M : \tau$ to term M jest silnie normalizowalny.
- Typ główny (najogólniejszy). Jeśli term M jest typizowalny, to istnieje dla niego typ główny, z którego można otrzymać wszystkie typy tego termu (przez odpowiednie podstawienia za zmienne przebiegające typy).

Definicja typów i funkcji dla spójników logicznych I

Zdefiniujemy w języku OCaml typy i funkcje odpowiadające spójnikom logicznym oraz ich regułom wprowadzania i eliminacji (patrz str. 30).

- Typy dla implikacji i koniunkcji są wbudowane. Implikacji odpowiada typ funkcyjny $\to$, a koniunkcji typ pary $*$, gdzie **pair** $x\, y \equiv (x, y)$.
- Alternatywa.

```
type ('a,'b) v = Inl of 'a | Inr of ('b);;
let case v p q =
   match v with
     Inl m -> p m
   | Inr m -> q m;;
val case : ('a, 'b) v -> ('a -> 'c) -> ('b -> 'c) -> 'c = <fun>
```

Definicja typów i funkcji dla spójników logicznych II

- Absurd.

```
type absurd;;
let absurdE (a:absurd) = failwith "absurdE";;
val absurdE : absurd -> 'a = <fun>
```

- Negacja.

```
type 'a not = Neg of ('a -> absurd);;
let negE (Neg m) n = m n;;
val negE : 'a not -> 'a -> absurd = <fun>
```

Analogicznie można te typy i funkcje zdefiniować w Haskellu. Zamiast alternatywy można użyć typu `Either` z modułu Prelude:

```
data Either a b = Left a | Right b
                  deriving (Eq, Ord, Read, Show)
```

Trzeba jednak wybrać opcję, umożliwiającą zdefiniowanie typu danych bez konstruktorów wartości (dla typu *absurd*), np. umieszczając na początku skryptu następujący wiersz:

```
 {-# LANGUAGE EmptyDataDecls #-}
```

$\vdash \neg\neg(p \vee \neg p)$

Uwaga. W logice konstruktywnej $\nvdash p \vee \neg p$.

$$\mathcal{D} = \dfrac{}{x : \neg(p \vee \neg p) \vdash x : \neg(p \vee \neg p)}\;(Ass)$$

$$\dfrac{\mathcal{D} \qquad \dfrac{\dfrac{\dfrac{\mathcal{D} \qquad \dfrac{\dfrac{}{y : p \vdash y : p}\;(Ass)}{y : p \vdash \mathbf{inl}\, y : p \vee \neg p}\;(\vee I)}{x : \neg(p \vee \neg p), y : p \vdash \mathbf{negE}\, x\, (\mathbf{inl}\, y) : \bot}\;(\neg E)}{x : \neg(p \vee \neg p) \vdash \mathbf{neg}\, \lambda y.\mathbf{negE}\, x\, (\mathbf{inl}\, y) : \neg p}\;(\neg I)}{x : \neg(p \vee \neg p) \vdash \mathbf{inr}\, (\mathbf{neg}\, \lambda y.\mathbf{negE}\, x\, (\mathbf{inl}\, y)) : p \vee \neg p}\;(\vee I)}{\dfrac{x : \neg(p \vee \neg p) \vdash \mathbf{negE}\, x\, (\mathbf{inr}\, (\mathbf{neg}\, \lambda y.\mathbf{negE}\, x\, (\mathbf{inl}\, y))) : \bot}{\vdash \mathbf{neg}\, \lambda x.\mathbf{negE}\, x\, (\mathbf{inr}\, (\mathbf{neg}\, \lambda y.\mathbf{negE}\, x\, (\mathbf{inl}\, y))) : \neg\neg(p \vee \neg p)}\;(\neg I)}\;(\neg E)$$

Oto odpowiedni program w języku OCaml:

```
let nnTND = Neg (fun x->negE x (Inr (Neg (fun y->negE x (Inl y)))));;
val nnTND : ('a, 'a not) v not not = Neg <fun>
```

Język programowania może być potraktowany jako stosowany rachunek λ, tzn. rachunek λ z dodanymi stałymi i regułami redukcji (ewaluacji) dla tych stałych.

Definicja

Zamknięty term M jest *zaklinowany* (ang.stuck) jeśli nie jest wartością, jednak nie istnieje term N taki, że $M \to_\delta N$.

Współczesne języki programowania są w większości *silnie typizowane* (ang. strongly typed, type safe), co oznacza, że poprawnie stypizowany term nie zaklinuje się. Jest to *poprawność* (ang. soundness) względem semantyki operacyjnej. Bardziej formalnie:

Definicja (Silna typizacja)

1. (Zachowanie typu) Jeśli $\Gamma \vdash M : \tau$ i $M \to_\delta N$ to $\Gamma \vdash N : \tau$.
2. (Progres) Jeśli $\vdash M : \tau$, to albo M jest wartością, albo istnieje term N taki, że $M \to_\delta N$.

let-polimorfizm, polimorfizm w stylu ML I

- *Polimorfizm* $\langle$gr. $\pi o\lambda \upsilon\varsigma$ = liczny + $\mu o\rho\varphi\eta$ = postać, forma$\rangle$ ogólnie oznacza wielopostaciowość i umożliwia przypisanie różnych typów temu samemu programowi. Istnieje wiele rodzajów polimorfizmu. Zwiększają one elastyczność języków programowania z typizacją statyczną.
- System $\lambda_{\rightarrow}$ w stylu Curry'ego ze zmiennymi przebiegającymi typy oferuje bardzo ograniczony rodzaj polimorfizmu. System jest klasyfikowany jako system z typami prostymi, ponieważ różne wystąpienia tej samej zmiennej w termie mają ten sam typ.

let-polimorfizm, polimorfizm w stylu ML II

- Możemy dodać do $\lambda_{\rightarrow}$ nową konstrukcję: **let** $x = N$ **in** M, gdzie $x \notin FV(N)$. W beztypowym rachunku lambda i w językach z typizacją dynamiczną (np. Scheme) ta konstrukcja może być traktowana jako lukier syntaktyczny dla $(\lambda x.M)N$. W językach z typizacją statyczną i typami prostymi jest to rzeczywiste rozszerzenie języka, ponieważ moduł typizacji pozwala na przypisanie różnym wystąpieniom w M wiązanej przez **let** zmiennej x różnych typów, co nie jest dozwolone w termie $\lambda x.M$. Wyrażenie **let** występuje w językach SML, OCaml i Haskell.
- W przykładzie poniżej pierwsze wyrażenie zostanie poprawnie stypizowane, natomiast drugie spowoduje błąd typu.

$$\textbf{let}\ x = \lambda y.y\ \textbf{in}\ xx \qquad (\lambda x.xx)(\lambda y.y)$$

let-polimorfizm, polimorfizm w stylu ML III

Potrzebna jest nowa reguła typizacji:

$$\frac{\Gamma \vdash M[x := N] : \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \textsf{\textbf{let}}\ x = N\ \textsf{\textbf{in}}\ M : \tau}\ (let)$$

i nowa reguła redukcji:

$$\textsf{\textbf{let}}\ x = N\ \textsf{\textbf{in}}\ M \to M[x := N]$$

- H.Barendregt, W.Dekkers, R.Statman, *Lambda Calculus with Types*, Cambridge University Press, Cambridge 2013
- H.P.Barendregt, *Lambda Calculi with Types*, w: S.Abramsky, Dov M. Gabbay, T.S.E. Maibaum, Handbook of Logic in Computer Science, vol. 2, Clarendon Press, Oxford 1992, pp. 117-309, http://www.cs.ru.nl/~henk/papers.html
- H.Barendregt, E.Barendsen, *Introduction to Lambda Calculus*, revised edition 1998, 2000, http://www.cse.chalmers.se/research/group/logic/TypesSS05/Extra/geuvers.pdf
- Ch.Hankin, *Lambda Calculi. A Guide for Computer Scientists*, Oxford University Press, Oxford 1994
- J.R.Hindley, *Basic Simple Type Theory*, Cambridge University Press, Cambridge 1997
- R.Nederpelt, H.Geuvers, *Type Theory and Formal Proof. An Introduction*, Cambridge University Press, Cambridge 2014
- M.H.Sørensen, P.Urzyczyn, *Lectures on the Curry-Howard Isomorphism*, Elsevier, Amsterdam 2006