System typów F_ω

Systemy Typów 2010/11
Prowadzący: dr Dariusz Biernacki

Piotr Polesiuk
`bassists@o2.pl`

Małgorzata Jurkiewicz
`gosia.jurkiewicz@gmail.com`

Wrocław, dnia 13 lutego 2011 r.

1. Wstęp

No to na razie taki bałagan

2. System F_ω

W rozdziale tym chcielibyśmy się skupić na systemie F_ω okrojonym do niezbędnego minimum. Przedstawimy, jak wyglądają termy, typy i wartości tego języka, a także pokażemy, jak przebiega typowanie, znajdowanie rodzaju, ewaluacja czy sprawdzanie równości typów. Postaramy się pisać jasno i pokażemy parę przykładów, aby nieobyty w temacie Czytelnik nie zgubił się. W następnych rozdziałach XXX-XXX do tak zdefiniowanego systemu będziemy wprowadzać rozszerzenia.

2.1. Termy i typy w F_ω

System F_ω to rachunek będący rozszerzeniem λ_ω oraz systemu F. Wszystkie trzy wywodzą się z rachunku lambda z typami prostymi. Termy oraz typy definiujemy w $\lambda_\to$ następująco:

$\mathtt{t} ::=$		*termy*
	$\mathtt{x}$	*zmienne*
	$\lambda\mathtt{x} : \mathtt{T}.\mathtt{t}$	*abstrakcja*
	$\mathtt{t}\ \mathtt{t}$	*aplikacja*
$\mathtt{T} ::=$		*typy*
	$\mathtt{X}$	*zmienna typowa*
	$\mathtt{T} \to \mathtt{T}$	*typ funkcji*

2.1.1. System λ_ω

Główną cechą systemu λ_ω jest to, że oprócz termów zależnych od termów mamy typy zależne od typów, czyli możemy mówić o aplikacji i abstrakcji typowej, a tak powstałe 'typy' będziemy nazywać konstruktorami. By nam się nie pomyliło z abstrakcją na termach, zmienne konstruktorowe będziemy zaczynać dużą literą. Przykładowo $\mathtt{Tb} = \lambda\mathtt{X}.\mathtt{X} \to \mathtt{Bool}$ i $\lambda\mathtt{X}.\mathtt{X}$ są abstrakcjami konstruktowymi, ale $\lambda\mathtt{x}.\mathtt{x}$ jest abstrakcją na termach. Do konstruktora $\mathtt{Tb}$ możemy zaaplikować $\mathtt{Bool}$ i dostaniemy $(\lambda\mathtt{X}.\mathtt{X} \to \mathtt{Bool})\mathtt{Bool}$ równoważne $\mathtt{Bool} \to \mathtt{Bool}$. Jak widać, użyliśmy słowa *równoważne*. W rachunku lambda z typami prostymi sposób konstrukcji typów gwarantował nam, że dwa typy $\mathtt{T_1}$ i $\mathtt{T_2}$ na pewno są różne (zakładając, że typy bazowe były sobie różne). W λ_ω jest inaczej – konstruktory tego systemu możemy podzielić na klasy równoważności. Do klasy $\mathtt{Bool} \to \mathtt{Bool}$ należą również $(\mathtt{Tb}^\mathtt{n})\mathtt{Bool}$ dla n naturalnego, a $\mathtt{T}^\mathtt{n}$ oznacza aplikację n konstruktorów $\mathtt{T}$. Zauważmy, że odpowiednikiem takiej relacji równoważności w $\lambda_\to$ jest β-równoważność. W świecie typów nazwiemy taką relację $\equiv$[1]. Każdy konstruktor typu jest silnie normalizowalny i zachodzi własność Churcha-Rossera. Przez $\mathtt{nf(T)}$ oznaczamy postać normalną konstruktora rodzaju $\mathtt{T}$. Dodatkowo wprowadzimy następującą regułę: $\frac{\Gamma\vdash\mathtt{t:S} \quad \mathtt{S}\equiv\mathtt{T}}{\Gamma\vdash\mathtt{t:T}}$ mówiącą, że jeżeli $\mathtt{S}$ jest konstruktorem termu $\mathtt{t}$, to dowolny konstruktor $\mathtt{S}$ równoważny z $\mathtt{T}$ również jest konstruktorem $\mathtt{t}$.

[1]formalnie zdefiniujemy tą relację w rozdziale XXX

Niestety, w tak zdefiniowanym systemie powstaje jeden problem. Nie chcielibyśmy, aby $\texttt{Bool}\,\texttt{Bool}$ było dozwolone, tak samo, jak w świecie termów nie chcieliśmy, by $\texttt{true}\,\texttt{true}$ było dozwolone. W świecie termów, by rozwiązać ten problem, wprowadziliśmy typy na termach, w świecie typów wprowadzimy *rodzaje* na konstruktorach. Piszemy, że $\texttt{T} :: \texttt{K}$, czyli konstruktor $\texttt{T}$ jest rodzaju $\texttt{K}$. Wprowadzimy też jeden rodzaj bazowy $*$.

Wszystkie typy, jakie pojawiły się w $\lambda_\to$, są rodzaju $*$. Np. $\texttt{Bool} :: *$, $\texttt{Nat} \to \texttt{Nat}$, $(\texttt{Bool} \to \texttt{Nat}) \to \texttt{Nat} :: *$, itd. Rodzaj $* \Rightarrow *$ będzie odpowiadał funkcjom z konstruktorów w konstruktory, np. $\lambda\texttt{X}.\texttt{X} \to \texttt{Bool} :: * \Rightarrow *$. $* \Rightarrow * \Rightarrow *$ bierze konstruktor i zwraca funkcję konstruktorową, np. $\lambda\texttt{X}.\lambda\texttt{Y}.\texttt{X} \to \texttt{Y} :: * \Rightarrow * \Rightarrow *$, itd.

Teraz możemy $\lambda_\to$ rozszerzyć o następujące konstrukcje:

- rodzaje

$\texttt{K} ::=$		*rodzaje*
	$*$	*rodzaj wszystkich typów*
	$\texttt{K} \Rightarrow \texttt{K}$	*rodzaj funkcji typowej*

- abstrakcję i aplikację typową na typach

$\texttt{T} ::=$	$\ldots$	*typy*
	$\lambda\texttt{X} :: \texttt{K}.\texttt{T}$	*abstrakcja konstruktorowa*
	$\texttt{T}\;\texttt{T}$	*aplikacji konstruktorowa*

Powstaje pytanie, czy wszystkie konstruktory są typami? Otóż nie, typy to konstruktory rodzaju $*$.

2.1.2. System *F*

System F jest systemem, w którym dodatkowo, oprócz termów zależnych od termów, mamy termy zależne od typów. Wprowadzimy trzeci już rodzaj abstrakcji i aplikacji, poprzedni był w świecie typów, ten będzie w świecie termów. Znana jest nam funkcja identycznościowa $\lambda\texttt{x}.\texttt{x}$, w $\lambda_\to$ możemy ją napisać na wiele sposób: $\lambda\texttt{x} : \texttt{Bool}.\texttt{x}$, $\lambda\texttt{x} : \texttt{Nat}.\texttt{x}$, $\lambda\texttt{x} : \texttt{Bool} \to \texttt{Nat}.\texttt{x}$. W systemie F możemy wszystkie te funkcje zapisać jako: $\lambda\texttt{X}.\lambda\texttt{x} : \texttt{X}.\texttt{x}$. Zauważmy, że ten *term* przyjmuje jako pierwszy argument typ, następnie term tego typu i zwraca term. Przykładem użycia takiego termu mogą być: $(\lambda\texttt{X}.\lambda\texttt{x} : \texttt{X}.\texttt{x})\;[\texttt{Bool}]\;\texttt{true}$, co daje $\texttt{true}$, albo $(\lambda\texttt{X}.\lambda\texttt{x} : \texttt{X}.\texttt{x})\;[\texttt{Nat}]\;\texttt{1}$, co daje $\texttt{1}$. W ten sposób powstała nam *uniwersalna* funkcja identycznościowa, której nadamy tzw. uniwersalny typ: $\lambda\texttt{X}.\lambda\texttt{x} : \texttt{X}.\texttt{x} \;:\; \forall\texttt{X}.\texttt{X} \to \texttt{X}$. Dodatkowo, jako że dodaliśmy już do systemu rodzaje, napiszemy $\lambda\texttt{X} :: *.\lambda\texttt{x} : \texttt{X}.\texttt{x} \;:\; \forall\texttt{X} :: *.\texttt{X} \to \texttt{X} \;::\; *$.

Czy moglibyśmy napisać $\lambda\texttt{X} :: * \Rightarrow *.\lambda\texttt{x} : \texttt{X}.\texttt{x} \;:\; \forall\texttt{X} :: * \Rightarrow *.\texttt{X} \to \texttt{X} \;::\; * \Rightarrow *$? Jak już mówiliśmy, tylko konstruktory rodzaju $*$ są typami, więc powyższy term nie jest dobry.

Po tym krótkim wstępie możemy już zdefiniować odziedziczone z systemu F własności takie, jak:

- abstrakcję i aplikację typową na termach

$\texttt{t} ::=$	$\ldots$	*termy*
	$\lambda\texttt{X} :: \texttt{K}.\texttt{t}$	*abstrakcja typowa*
	$\texttt{t}[\texttt{T}]$	*aplikacja typowa*

- typ uniwersalny

$\mathtt{T} ::=$	$\ldots$	*typy*
	$\forall \mathtt{X} :: \mathtt{K}.\mathtt{T}$	*typ uniwersalny*

2.2. Typowanie

2.2.1. Kontekst

Kontekst typowania opisany jest następującą składnią abstrakcyjną:

$\Gamma ::=$		*kontekst*
	$\emptyset$	*pusty kontekst*
	$\Gamma, \mathtt{x} : \mathtt{T}$	*wiązanie typu*
	$\Gamma, \mathtt{X} :: \mathtt{K}$	*wiązanie rodzaju*

Konteksty typowania bedziemy często traktować jako skończone zbiory wiązań i będziemy używać teoriomnogościowych symboli na nich. Np. przynależność do kontekstu formalnie definiujemy jako:

$$\frac{}{B \in \Gamma, B} \qquad \frac{B \in \Gamma}{B \in \Gamma, B'}$$

Definicje pozostałych operacji teoriomnogościowych są na tyle naturalne, że zostawiamy je Czytelnikowi do uzupełnienia.

2.2.2. Podstawienia

Oprócz zwykłego podstawienia za zmienne, które pozostawiamy Czytelnikowi do uzupełnienia, powinniśmy zdefiniować podstawienie za zmienne konstruktorowe.

- $[\mathtt{Y} \mapsto \mathtt{T}]\mathtt{X} = \begin{cases} \mathtt{T} & Y = X \\ \mathtt{X} & \text{w.p.p} \end{cases}$
- $[\mathtt{Y} \mapsto \mathtt{T}](\mathtt{X}_1\ \mathtt{X}_2) = [\mathtt{Y} \mapsto \mathtt{T}]\mathtt{X}_1[\mathtt{Y} \mapsto \mathtt{T}]\mathtt{X}_2$
- $[\mathtt{Y} \mapsto \mathtt{T}](\mathtt{S}_1 \to \mathtt{S}_2) = [\mathtt{Y} \mapsto \mathtt{T}]\mathtt{S}_1 \to [\mathtt{Y} \mapsto \mathtt{T}]\mathtt{S}_2$
- $[\mathtt{Y} \mapsto \mathtt{T}]\forall \mathtt{X}.\mathtt{S} = \begin{cases} \forall \mathtt{X}.\mathtt{S} & Y = X \text{ lub } Y \notin FV(S) \\ \forall \mathtt{X}.[\mathtt{Y} \mapsto \mathtt{T}]\mathtt{S} & X \notin FV(S) \text{ i } Y \in FV(S) \end{cases}$
- $[\mathtt{Y} := \mathtt{T}]\lambda \mathtt{X}.\mathtt{S} = \begin{cases} \lambda \mathtt{X}.\mathtt{S} & Y = X \text{ lub } Y \notin FV(S) \\ \lambda \mathtt{X}.[\mathtt{Y} \mapsto \mathtt{T}]\mathtt{S} & X \notin FV(S) \text{ i } Y \in FV(S) \end{cases}$

2.2.3. Relacja $\equiv$

Jak wspomnieliśmy w rozdziale XXX, definiujemy na typach relację równoważności. W poniższych wzorach $\mathtt{S}, \mathtt{S}_1, \mathtt{S}_2, \mathtt{T}, \mathtt{T}_1, \mathtt{T}_2$ to typy, $\mathtt{K}$ to rodzaj. Następujące trzy reguły:

$$\frac{}{\mathtt{T} \equiv \mathtt{T}} \qquad \frac{\mathtt{S} \equiv \mathtt{T}}{\mathtt{T} \equiv \mathtt{S}} \qquad \frac{S \equiv U \quad U \equiv T}{S \equiv T}$$

gwarantują nam równoważność relacji $\equiv$. Pozostałe reguły jak następuje:

$$\frac{\mathtt{S}_1 \equiv \mathtt{T}_1 \quad \mathtt{S}_2 \equiv \mathtt{T}_2}{\mathtt{S}_1 \to \mathtt{S}_2 \equiv \mathtt{T}_1 \to \mathtt{T}_2} \qquad \frac{\mathtt{S}_1 \equiv \mathtt{T}_1 \quad \mathtt{S}_2 \equiv \mathtt{T}_2}{\mathtt{S}_1\ \mathtt{S}_2 \equiv \mathtt{T}_1\ \mathtt{T}_2}$$

$$\frac{\mathtt{S} \equiv \mathtt{T}}{\lambda\mathtt{X} :: \mathtt{K}.\mathtt{S} \equiv \lambda\mathtt{X} :: \mathtt{K}.\mathtt{T}} \qquad (\lambda\mathtt{X} :: \mathtt{K}.\mathtt{S})\mathtt{T} \equiv [\mathtt{X} \mapsto \mathtt{T}]\mathtt{S}$$

definiują równoważność funkcji typowych, aplikacji i abstrakcji konstruktorowych oraz typów uniwersalnych.

2.2.4. Reguły znajdowania rodzaju

W systemie F_ω każdemu poprawnie zbudowanemu typowi przyporządkowujemy rodzaj. Przyporządkowanie to określa relacja $(. \vdash . :: .)$ zdefiniowana następująco.

Jeżeli zachodzi $\Gamma \vdash \mathtt{T} :: \mathtt{K}$, to powiemy, że *typ* $\mathtt{T}$ *jest rodzaju* $\mathtt{K}$ *w kontekście* Γ, gdzie relacja określenia rodzaju $(. \vdash . :: .) \subseteq \Gamma \times T \times K$ jest najmniejszą relacją zamkniętą na reguły:

$$\frac{\mathtt{X} :: \mathtt{K} \in \Gamma}{\Gamma \vdash \mathtt{X} :: \mathtt{K}} \qquad \frac{\Gamma \vdash \mathtt{T}_1 :: \mathtt{K}_1 \Rightarrow \mathtt{K}_2 \quad \Gamma \vdash \mathtt{T}_2 :: \mathtt{K}_1}{\Gamma \vdash \mathtt{T}_1\mathtt{T}_2 :: \mathtt{K}_2}$$

$$\frac{\Gamma \vdash \mathtt{X} :: \mathtt{K}_1 \quad \Gamma \vdash \mathtt{T} :: \mathtt{K}_2}{\Gamma \vdash \lambda\mathtt{X} :: \mathtt{K}_1.\mathtt{T} :: \mathtt{K}_1 \Rightarrow \mathtt{K}_2} \qquad \frac{\Gamma \vdash \mathtt{X} :: \mathtt{K} \quad \Gamma \vdash \mathtt{T} :: *}{\Gamma \vdash \forall\mathtt{X} :: \mathtt{K}.\mathtt{T} :: *}$$

$$\frac{\Gamma \vdash \mathtt{T}_1 : * \quad \Gamma \vdash \mathtt{T}_2 : *}{\Gamma \vdash \mathtt{T}_1 \to \mathtt{T}_2 : *}$$

2.2.5. Reguły typowania

Jesteśmy już gotowi przedstawić reguły typowania zdefiniowanego wyżej systemu F_ω. Każdemu poprawnie zbudowanemu termowi przyporządkowujemy typ. Przyporządkowanie to określa relacja $(. \vdash . : .)$ zdefiniowana następująco.

$$\frac{\mathtt{x} : \mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \qquad \frac{\Gamma \vdash \mathtt{T}_1 :: * \quad \Gamma, \mathtt{x} : \mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda\mathtt{x} : \mathtt{T}_1.\mathtt{t}_2 \ : \ \mathtt{T}_1 \to \mathtt{T}_2}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \to \mathtt{T}_2 \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_1}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_2} \qquad \frac{\Gamma \vdash \mathtt{t} : \mathtt{S} \quad \mathtt{S} \equiv \mathtt{T} \quad \Gamma \vdash \mathtt{T} :: *}{\Gamma \vdash \mathtt{t} : \mathtt{T}}$$

$$\frac{\Gamma, \mathtt{X} :: \mathtt{K} \vdash \mathtt{t} : \mathtt{T}}{\Gamma \vdash \lambda\mathtt{X} :: \mathtt{K}.\mathtt{t} \ : \forall\mathtt{X} :: \mathtt{K}.\mathtt{T}} \qquad \frac{\Gamma \vdash \mathtt{t} : \forall\mathtt{X} :: \mathtt{K}.\mathtt{T} \quad \Gamma \vdash \mathtt{T}' :: \mathtt{K}}{\Gamma \vdash \mathtt{t}[\mathtt{T}'] \ : [\mathtt{X} \mapsto \mathtt{T}']\mathtt{T}}$$

2.3. Ewaluacja

Wartości w F_ω zdefiniujemy dokładnie jak w $\lambda_\to$.

$\mathtt{v}$ ::=		*wartości*
	$\lambda\mathtt{x} : \mathtt{T}.\mathtt{t}$	*wartość abstrakcji*

Ewaluacja przebiega w sposób standardowy dla aplikacji i abstrakcji termów. Teraz, dla czytelności, przetoczymy te reguły ewaluacji ($\mathtt{t}_1, \mathtt{t}_1', \mathtt{t}_2, \mathtt{t}_2', \mathtt{t}$ to termy, $\mathtt{v}$ to wartość, $\mathtt{x}{:}\mathtt{T}$ to zmienna $\mathtt{x}$ typu $\mathtt{T}$):

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \qquad \frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'}$$

$$(\lambda \mathtt{x} : \mathtt{T}.\mathtt{t})\mathtt{v} \longrightarrow [\mathtt{x} \mapsto \mathtt{v}]\mathtt{t}$$

Do tego dochodzą reguły dla nowych w języku abstrakcji typowych i aplikacji typowych.

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{t}[\mathtt{T}] \longrightarrow \mathtt{t}'[\mathtt{T}]}$$

$$(\lambda \mathtt{X} :: \mathtt{K}.\mathtt{t})[\mathtt{T}] \longrightarrow [\mathtt{X} \mapsto \mathtt{T}]\mathtt{t}$$

3. Rozszerzenia F_ω

Ważne!!! Zaoszczędza dużo pisania i pozwala w zasanie na przepisanie regułek z innych systemów ;)
Typowanie:

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{T} \quad \mathtt{S} \equiv \mathtt{T} \quad \Gamma \vdash \mathtt{S} :: *}{\Gamma \vdash \mathtt{t} : \mathtt{S}}$$

3.1. wyrażenia arytmetyczne i logiczne

t ::=	...	*termy*
	true	*prawda*
	false	*fałsz*
	zero	*zero*
	succ t	*następnik*
	pred t	*poprzednik*
	iszero	*test na zero*
	if t then t else t	*warunek*
T ::=	...	*typy*
	Nat	*typ liczbowy*
	Bool	*typ boolowski*
v ::=	...	*typy*
	true	*wartość prawdy*
	false	*wartość fałszu*
	nv	*wartość liczbowa*
nv ::=	...	*wartość liczbowa*
	zero	*wartość zera*
	succ nv	*wartość następnika*

Tworzenie rodzaju:

$$\frac{}{\Gamma \vdash \mathtt{Bool} :: *} \qquad \frac{}{\Gamma \vdash \mathtt{Nat} :: *}$$

Typowanie:

$$\frac{}{\Gamma \vdash \mathtt{true} : \mathtt{Bool}} \qquad \frac{}{\Gamma \vdash \mathtt{false} : \mathtt{Bool}} \qquad \frac{\Gamma \vdash \mathtt{t} : \mathtt{Nat}}{\Gamma \vdash \mathtt{iszero}\ \mathtt{t} : \mathtt{Bool}}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \quad \Gamma \vdash \mathtt{t}_3 : \mathtt{T} \quad \Gamma \vdash \mathtt{T} :: *}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}}$$

$$\frac{}{\Gamma \vdash \texttt{zero} : \texttt{Nat}} \qquad \frac{\Gamma \vdash \texttt{t} : \texttt{Nat}}{\Gamma \vdash \texttt{succ t} : \texttt{Nat}} \qquad \frac{\Gamma \vdash \texttt{t} : \texttt{Nat}}{\Gamma \vdash \texttt{pred t} : \texttt{Nat}}$$

Ewaluacja:

$$\texttt{aaa}$$

3.2. unit i sekwencje

W zasadzie unita można nie wprowadzać:
$\texttt{unit} = \lambda\texttt{X} :: *.\lambda\texttt{x} : \texttt{X}.\texttt{x}$
$\texttt{Unit} = \forall\texttt{X} :: *.\texttt{X} \to \texttt{X}$
Typowanie:

$$\frac{}{\Gamma \vdash \texttt{unit} : \texttt{Unit}}$$

$$\texttt{t}_1; \texttt{t}_2 = (\lambda\texttt{x} : \texttt{Unit}.\texttt{t}_2)\texttt{t}_1 \qquad \text{gdzie x} \notin \text{FV}(t_2)$$

a to stare, może się przyda:

$\texttt{t} ::=$	...	*termy*
	$\texttt{unit}$	
$\texttt{T} ::=$	...	*typy*
	$\texttt{Unit}$	
$\texttt{v} ::=$	...	*wartości*
	$\texttt{unit}$	

Tworzenie rodzaju:

$$\frac{}{\Gamma \vdash \texttt{Unit} :: *}$$

Typowanie:

$$\frac{}{\Gamma \vdash \texttt{unit} : \texttt{Unit}}$$

$$\texttt{t}_1; \texttt{t}_2 = (\lambda\texttt{x} : \texttt{Unit}.\texttt{t}_2)\texttt{t}_1 \qquad \text{gdzie x} \notin \text{FV}(t_2)$$

Ewaluacja:

$$\texttt{aaa}$$

3.3. Definicje lokalne

$$\texttt{let x} = \texttt{t}_1 \texttt{ in t}_2 \overset{\texttt{def}}{=} (\lambda\texttt{x} : \texttt{T}.\texttt{t}_2)\texttt{t}_1 \qquad \text{gdzie } \texttt{t}_1 : \texttt{T}$$

3.4. Rekordy

$\texttt{t} ::=$	...	*termy*
	$\{\texttt{l}_\texttt{i} = \texttt{t}_\texttt{i}\ ^{\texttt{i}\in 1..\texttt{n}}\}$	*rekord*
	$\texttt{t.l}$	*projekcja*
$\texttt{T} ::=$	...	*typy*
	$\{\texttt{l}_\texttt{i} : \texttt{T}_\texttt{i}\ ^{\texttt{i}\in 1..\texttt{n}}\}$	*typ rekordu*
$\texttt{v} ::=$	...	*wartości*
	$\{\texttt{l}_\texttt{i} = \texttt{v}_\texttt{i}\ ^{\texttt{i}\in 1..\texttt{n}}\}$	*wartość rekordu*

Tworzenie rodzaju:

$$\frac{\Gamma \vdash \mathtt{T}_1 :: * \ \dots \ \Gamma \vdash \mathtt{T}_n :: *}{\Gamma \vdash \{\mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n}\} :: *}$$

Typowanie:

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \ \dots \ \Gamma \vdash \mathtt{t}_n : \mathtt{T}_n \quad \Gamma \vdash \{\mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n}\} :: *}{\Gamma \vdash \{\mathtt{l_i} = \mathtt{t_i}^{\ i \in 1..n}\} : \{\mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n}\}}$$

$$\frac{\Gamma \vdash \mathtt{t} : \{\mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n}\} \quad \Gamma \vdash \{\mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n}\} :: *}{\Gamma \vdash \mathtt{t.i} : \mathtt{T_i}}$$

Ewaluacja:

$$\{\mathtt{l_i} = \mathtt{v_i}^{\ i \in 1..n}\}.\mathtt{i} \longrightarrow \mathtt{v_i} \qquad \frac{\mathsf{e} \longrightarrow \mathsf{e}'}{\mathsf{e.i} \longrightarrow \mathsf{e}'.\mathsf{i}}$$

$$\frac{\mathtt{t_i} \longrightarrow \mathtt{t}'_i}{\{\mathtt{l}_1 = \mathtt{v}_1, \dots, \mathtt{l}_{i-1} = \mathtt{v}_{i-1}, \mathtt{l_i} = \mathtt{t_i}, \dots, \mathtt{l_n} = \mathtt{t_n}\} \longrightarrow \{\mathtt{l}_1 = \mathtt{v}_1, \dots, \mathtt{l}_{i-1} = \mathtt{v}_{i-1}, \mathtt{l_i} = \mathtt{t}'_i, \dots, \mathtt{l_n} = \mathtt{t_n}\}}$$

3.5. Warianty

$\mathtt{t} ::=$	$\dots$	*termy*
	$< \mathtt{l} = \mathtt{t} > \ \mathsf{as}\ \mathtt{T}$	*tagowanie*
	$\mathsf{case}\ \mathtt{t}\ \mathsf{of}\ < \mathtt{l_i} = \mathtt{x_i} > \Rightarrow \mathtt{t_i}^{\ i \in 1..n}$	*case*
$\mathtt{T} ::=$	$\dots$	*typy*
	$< \mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n} >$	*typ wariantu*

Tworzenie rodzaju:

$$\frac{\Gamma \vdash \mathtt{T}_1 :: * \ \dots \ \Gamma \vdash \mathtt{T}_n :: *}{\Gamma \vdash < \mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n} > :: *}$$

Typowanie:

$$\frac{\Gamma \vdash \mathtt{t}_0 : < \mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n} > \qquad \Gamma, \mathtt{x}_1 : \mathtt{T}_1 \vdash \mathtt{t}_1 : \mathtt{T} \ \dots \ \Gamma, \mathtt{x_n} : \mathtt{T_n} \vdash \mathtt{t_n} : \mathtt{T}}{\Gamma \vdash \mathsf{case}\ \mathtt{t}_0\ \mathsf{of}\ < \mathtt{l_i} = \mathtt{x_i} > \Rightarrow \mathtt{t_i}^{\ i \in 1..n} : \mathtt{T}}$$

$$\frac{\Gamma \vdash \mathtt{t_j} : \mathtt{T_j} \qquad \Gamma \vdash < \mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n} > :: *}{\Gamma \vdash < \mathtt{l_j} = \mathtt{t_j} > \mathsf{as} < \mathtt{l_i} : \mathtt{T_i}^{\ i \in 1..n} > : < \mathtt{l_i} : \ \mathtt{T_i}^{\ i \in 1..n} >}$$

Ewaluacja:

$$\mathsf{aaa}$$

3.6. Punkt stały

$\mathtt{t} ::=$	$\dots$	*termy*
	$\mathtt{fix}\ \mathtt{t.v}$	*punkt stały*

Typowanie

$$\frac{\Gamma, \mathtt{f} : \mathtt{T} \vdash \mathtt{v} : \mathtt{S} \qquad \Gamma \vdash \mathtt{T} :: * \qquad \Gamma \vdash \mathtt{S} :: * \qquad \mathtt{S} \equiv \mathtt{T}}{\Gamma \vdash \mathtt{fix}\ \mathtt{f.v} : \mathtt{T}}$$

Ewaluacja

$$\mathtt{fix}\ \mathtt{f.v} \longrightarrow [\mathtt{f} \mapsto \mathtt{fix}\ \mathtt{f.v}]\mathtt{v}$$

3.7. listy

Jako przykład wbudowanych typów danych wybraliśmy listy. Podobne rekursywne struktury, jak na przykład drzewa, możemy dodać do języka w analogiczny sposób, jednak rekurencyjne typy danych odwiodą nas od tej konieczności.

$\mathtt{t} ::=$	$\ldots$	*termy*
	$\mathtt{nil}[\mathtt{T}]$	*lista pusta*
	$\mathtt{cons}[\mathtt{T}]\ \mathtt{t}\ \mathtt{t}$	*konstruktor listy*
	$\mathtt{isnil}[\mathtt{T}]\ \mathtt{t}$	*test na pustość listy*
	$\mathtt{head}[\mathtt{T}]\ \mathtt{t}\ \mathtt{t}$	*głowa listy*
	$\mathtt{tail}[\mathtt{T}]\ \mathtt{t}$	*ogon listy*
$\mathtt{T} ::=$		*typy*
	$\mathtt{List}\ \mathtt{T}$	*typ listy*
$\mathtt{v} ::=$		*typy*
	$\mathtt{nil}\ [\mathtt{T}]$	*wartość pustej listy*
	$\mathtt{cons}\ [\mathtt{T}]\ \mathtt{v}\ \mathtt{v}$	*wartość listy niepustej*

Tworzenie rodzaju:

$$\frac{\Gamma \vdash \mathtt{T} :: *}{\Gamma \vdash \mathtt{List}\ \mathtt{T} :: *}$$

Typowanie:

$$\frac{\Gamma \vdash \mathtt{List}\ \mathtt{T} :: *}{\Gamma \vdash \mathtt{nil}[\mathtt{T}] : \mathtt{List}\ \mathtt{T}} \qquad \frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T} \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{List}\ \mathtt{T}}{\Gamma \vdash \mathtt{List}[\mathtt{T}]\ \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{List}\ \mathtt{T}}$$

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{List}\ \mathtt{T}}{\Gamma \vdash \mathtt{head}[\mathtt{T}]\ \mathtt{t} : \mathtt{T}} \qquad \frac{\Gamma \vdash \mathtt{t} : \mathtt{List}\ \mathtt{T}}{\Gamma \vdash \mathtt{tail}[\mathtt{T}]\ \mathtt{t} : \mathtt{List}\ \mathtt{T}}$$

Ewaluacja:

$$\mathtt{aaa}$$

3.8. Typy egzystencjalne

System F_ω jest już w stanie zakodować typy egzystencjalne, choć wbudowane typy egzystencjalne niczemu nie szkodzą. Pokażemy oba podejścia do tego problemu, zaczynając od przedstawienia składni:

$\mathtt{t} ::=$	$\cdots$	*termy*
	$\{{}^*\mathtt{T} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as}\ \mathtt{T}$	*pakowanie*
	$\mathtt{let}\ \{\mathtt{X}, \mathtt{x}\} = \mathtt{t}\ \mathtt{in}\ \mathtt{t}$	*odpakowanie*
$\mathtt{T} ::=$	$\cdots$	*typy*
	$\{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$	*typ egzystencjalny*
$\mathtt{v} ::=$	$\cdots$	*wartości*
	$\{{}^*\mathtt{T}, \mathtt{v}\}\ \mathtt{as}\ \mathtt{T}$	*pakowanie*

W systemie F_ω powyższe elementy języka możemy zdefiniować następująco:

$\{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\} \stackrel{\text{def}}{=} \forall \mathtt{Y} :: *.(\forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}) \to \mathtt{Y}$

$\{{}^*\mathtt{U} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as}\ \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\} \stackrel{\text{def}}{=} \mathtt{let}\ \mathtt{x} = \mathtt{t}\ \mathtt{in}\ \lambda \mathtt{Y} :: *.(\lambda \mathtt{f} : \forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}).\mathtt{f}\ [\mathtt{U}]\ \mathtt{x}$

$\mathtt{let}\ \{\mathtt{X} :: \mathtt{K}, \mathtt{x}\} = \mathtt{t}\ \mathtt{in}\ \mathtt{t}' \stackrel{\text{def}}{=} \mathtt{t}[\mathtt{T}'](\lambda \mathtt{X} :: \mathtt{K}.\lambda \mathtt{x} : \mathtt{T}.\mathtt{t}') \qquad$ gdzie $\mathtt{t}' : \mathtt{T}'$

Zauważmy, że dopiero obecność rodzajów pozwoliła nam na tego rodzaju sztuczki. W systemie F nie umiemy tak zrobić.

Na pierwszy rzut oka termy te są niezrozumiałe. Ależ jak bardzo można się mylić – są miłe i przyjemne dla swych wielbicieli. Pokażemy, że zachodzą podstawowe własności pakowania i odpakowania. Rozważmy term $\{^*\mathtt{U} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as}\ \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$.

$\{^*\mathtt{U} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as}\ \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$
$= \mathtt{let\ x = t\ in}\ \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K.T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}]\mathtt{x} =$
$= (\lambda \mathtt{x} : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{T}.\lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K.T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}]\mathtt{x})\mathtt{t} =$
$\overset{\mathtt{t}:[\mathtt{X}\mapsto\mathtt{U}]\mathtt{T}}{=} \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K.T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}](\mathtt{t} : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{T})$
co jest typu $\forall \mathtt{Y} :: *.(\forall \mathtt{X} :: \mathtt{K.T} \to \mathtt{Y}) \to \mathtt{Y}$, czyli z definicji $\{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$.

Rozważmy bardziej życiowy przykład, aby Czytelnik mógł jeszcze raz przeanalizować pakowanie. Oto typowanie w systemie F przykładowego termu:

$$\frac{\Gamma \vdash \{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\} : [\mathtt{X} \mapsto \mathtt{Nat}]\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\}}{\Gamma\{^*\mathtt{X}, \{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\}\}\ \mathtt{as}\ \{\exists \mathtt{X}, \{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\}\}}$$

Następnie wyprowadzimy ten term w F_ω:

$\{\mathtt{Nat} :: \mathtt{K}, \{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\}\}\ \mathtt{as}\ \{\exists \mathtt{X} :: \mathtt{K}, \{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\}\} =$
$= \mathtt{let\ x} = \{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\}\ \mathtt{in}\ \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\} \to \mathtt{Y}).\mathtt{f}[\mathtt{Nat}]\mathtt{x} =$
$= (\lambda \mathtt{x} : [\mathtt{X} \mapsto \mathtt{Nat}]\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\}.\lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\} \to \mathtt{Y}).\mathtt{f}[\mathtt{Nat}]\mathtt{x})$
$\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\} \overset{\{\mathtt{a}=\mathtt{zero},\mathtt{f}:\lambda\mathtt{x}:\mathtt{Nat.succ\ x}\}:[\mathtt{X}\mapsto\mathtt{Nat}]\{\mathtt{a}:\mathtt{X},\mathtt{f}:\mathtt{X}\to\mathtt{Nat}\}}{=}$
$= \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\} \to \mathtt{Y}).\mathtt{f}[\mathtt{Nat}]\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\}$

co jest typu $\forall \mathtt{Y} :: *.(\forall \mathtt{X} :: \mathtt{K}.\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\} \to \mathtt{Y}) \to \mathtt{Y}$, czyli z definicji $\{\exists \mathtt{X} :: \mathtt{K}, \{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\}\}$.

Uważne odpakowanie otrzymanego termu pozostawiamy Czytelnikowi jako ćwiczenie, my pozwolimy sobie przeprowadzać schemat wywodu:
$\mathtt{let}\ \{\mathtt{X}, \mathtt{x}\} = \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\} \to \mathtt{Y}).\mathtt{f}[\mathtt{Nat}]\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\}\ \mathtt{in}\ (\mathtt{x.f\ x.a}) =$
$= (\lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\} \to \mathtt{Y}).\mathtt{f}[\mathtt{Nat}]\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\})[\mathtt{T}'](\lambda \mathtt{X} :: \mathtt{K}.\lambda \mathtt{x} : \mathtt{T}.(\mathtt{x.f\ x.a})) =$
$= ((\lambda \mathtt{X} :: \mathtt{K}.\lambda \mathtt{x} : \mathtt{T}.(\mathtt{x.f\ x.a}))[\mathtt{Nat}](\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\} : [\mathtt{X} \mapsto \mathtt{Nat}]\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\})) =$
$= (\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\}.\mathtt{f}\ \{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat.succ\ x}\}.\mathtt{a}) =$
$= (\lambda \mathtt{x} : \mathtt{Nat.succ\ x})\mathtt{zero} = \mathtt{succ\ zero}$

Przykłady powyższe obrazują działanie zakodowanych typów rekurencyjnych. Teraz zdefiniujemy wbudowane w język konstrukcje typów rekurencyjnych dla systemu F_ω. Do definicji termów, typów i wartości dodaliśmy już elementy w tabelce na początku rozdziału. Pokażemy, w jaki sposób przebiega typowanie i ewaluacja.

Tworzenie rodzaju:

$$\frac{\mathtt{a\ tutaj\ co?}}{\Gamma \vdash \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\} :: \mathtt{K}}$$

Typowanie:

$$\frac{\Gamma \vdash \mathtt{t} : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{T} \qquad \Gamma \vdash \mathtt{U} :: \mathtt{K} \qquad \Gamma \vdash \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\} :: *}{\Gamma \vdash \{^*\mathtt{U} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as}\ \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}\ :\ \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}_1\} \qquad \Gamma, \mathtt{X} :: \mathtt{K}, \mathtt{x} : \mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \mathtt{let}\ \{\mathtt{X}, \mathtt{x}\} = \mathtt{t}_1\ \mathtt{in}\ \mathtt{t}_2 : \mathtt{T}_2}$$

Ewaluacja:

$$\mathtt{let}\ \{\mathtt{X}, \mathtt{x}\} = (\{^*\mathtt{U} :: \mathtt{K}, \mathtt{v}\}\ \mathtt{as\ T})\ \ \mathtt{in\ t} \longrightarrow [\mathtt{X} \mapsto \mathtt{U}][\mathtt{x} \mapsto \mathtt{v}]\mathtt{t}$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\{^*\mathtt{U} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as\ T} \longrightarrow \{^*\mathtt{U} :: \mathtt{K}, \mathtt{t}'\}\ \mathtt{as\ T}}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{let}\ \{\mathtt{X},\mathtt{x}\} = \mathtt{t_1}\ \ \mathtt{in}\ \mathtt{t_2} \longrightarrow \mathtt{let}\ \{\mathtt{X},\mathtt{x}\} = \mathtt{t_1'}\ \ \mathtt{in}\ \mathtt{t_2}}$$

3.9. Typy rekurencyjne

Hmmm, no tu musze się zastanowić, to na dole dla zwykłej wersji.

$\mathtt{t}$ ::=	...	*termy*
	$\mathtt{fold[T]\ t}$	*folding*
	$\mathtt{unfold[T]\ t}$	*unfolding*
$\mathtt{T}$ ::=	...	*typy*
	$\mu\mathtt{X.T}$	*typ rekursywny*
$\mathtt{v}$ ::=	...	*wartości*
	$\mathtt{fold[T]\ v}$	*folding*

Tworzenie rodzaju:

$$\frac{}{\Gamma \vdash \mu\mathtt{X.T} :: *}$$

Typowanie:

$$\frac{\mathtt{U} = \mu\mathtt{X.T} \qquad \Gamma \vdash \mathtt{t} : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{U}}{\Gamma \vdash \mathtt{fold[U]\ t}\ : \mathtt{U}}$$

Ewaluacja:

3.10. dopasowanie wzorca

4. Sładnia abstrakcyjna języka

5. Semantyka i typowanie

6. Rekonstrukcja typów

7. Własności i dowody

7.1. Inne własności F_ω

Definicja 1. Reguły przepisywania typów w systemie F_ω w wersji Curry'ego standardowe, oprócz:

$\frac{\Gamma \vdash M : \forall X \sigma}{\Gamma \vdash M : nf(\sigma[X := \tau])}$

Nierozstzygalne są problemy:

- sprawdzania typu: dane Γ, M, τ, Czy $\Gamma \vdash M : \tau$
- typowalność: dane M, Czy $\exists \Gamma \tau . \Gamma \vdash M : \tau$

7.2. pare słów o rozszerzeniach

8. Praktyczne zastosowanie

9. Podsumowanie

Literatura

[1] Pierce,