

# System typów $F_\omega$

Systemy Typów 2010/11  
Prowadzący: dr Dariusz Biernacki

Piotr Polesiuk	Małgorzata Jurkiewicz
bassists@o2.pl	gosia.jurkiewicz@gmail.com

Wrocław, dnia 13 lutego 2011 r.

## 1. Wstęp

No to na razie taki bałagan

## 2. System $F_\omega$

W rozdziale tym chcielibyśmy się skupić na systemie  $F_\omega$  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 $F_\omega$

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

$t ::=$	<i>termy</i>
$x$	<i>zmienne</i>
$\lambda x : T. t$	<i>abstrakcja</i>
$t t$	<i>aplikacja</i>
$T ::=$	<i>typy</i>
$X$	<i>zmienna typowa</i>
$T \rightarrow T$	<i>typ funkcji</i>

#### 2.1.1. System $\lambda_\omega$

Główną cechą systemu  $\lambda_\omega$  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  $Tb = \lambda X. X \rightarrow Bool$  i  $\lambda X. X$  są abstrakcjami konstruktorowymi, ale  $\lambda x. x$  jest abstrakcją na termach. Do konstruktora  $Tb$  możemy zaaplikować  $Bool$  i dostaniemy  $(\lambda X. X \rightarrow Bool)Bool$  równoważne  $Bool \rightarrow Bool$ . Jak widać, użyliśmy słowa *równoważne*. W rachunku lambda z typami prostymi sposób konstrukcji typów gwarantował nam, że dwa typy  $T_1$  i  $T_2$  na pewno są różne (zakładając, że typy bazowe były sobie różne). W  $\lambda_\omega$  jest inaczej – konstruktory tego systemu możemy podzielić na klasy równoważności. Do klasy  $Bool \rightarrow Bool$  należą również  $(Tb^n)Bool$  dla  $n$  naturalnego, a  $T^n$  oznacza aplikację  $n$  konstruktorów  $T$ . Zauważmy, że odpowiednikiem takiej relacji równoważności w  $\lambda_\rightarrow$  jest  $\beta$ -równoważność. W świecie typów nazwiemy taką relację  $\equiv^1$ . Każdy konstruktor typu jest silnie normalizowalny i zachodzi własność Churcha-Rossera. Przez  $nf(T)$  oznaczamy postać normalną konstruktora rodzaju  $T$ . Dodatkowo wprowadzimy następującą regułę:  $\frac{\Gamma \vdash t : S \quad S \equiv T}{\Gamma \vdash t : T}$  mówiącą, że jeżeli  $S$  jest konstruktorem termu  $t$ , to dowolny konstruktor  $S$  równoważny z  $T$  również jest konstruktorem  $t$ .

<sup>1</sup>formalnie zdefiniujemy tą relację w rozdziale XXX

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

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

Teraz możemy  $\lambda_{\rightarrow}$  rozszerzyć o następujące konstrukcje:

- rodzaje

$K ::=$	$*$	<i>rodzaje</i>
	$K \Rightarrow K$	<i>rodzaj wszystkich typów</i>
		<i>rodzaj funkcji typowej</i>

- abstrakcję i aplikację typową na typach

$T ::=$	$\dots$	<i>typy</i>
	$\lambda X :: K.T$	<i>abstrakcja konstruktorowa</i>
	$T T$	<i>aplikacji konstruktorowa</i>

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  $\lambda x.x$ , w  $\lambda_{\rightarrow}$  możemy ją napisać na wiele sposobów:  $\lambda x : Bool.x$ ,  $\lambda x : Nat.x$ ,  $\lambda x : Bool \rightarrow Nat.x$ . W systemie  $F$  możemy wszystkie te funkcje zapisać jako:  $\lambda X.\lambda x : X.x$ . Zauważmy, że ten *term* przyjmuje jako pierwszy argument typ, następnie term tego typu i zwraca term. Przykładem użycia takiego termu mogą być:  $(\lambda X.\lambda x : X.x) [Bool] \text{ true}$ , co daje `true`, albo  $(\lambda X.\lambda x : X.x) [Nat] 1$ , co daje `1`. W ten sposób powstała nam *uniwersalna* funkcja identycznościowa, której nadamy tzw. uniwersalny typ:  $\lambda X.\lambda x : X.x : \forall X.X \rightarrow X$ . Dodatkowo, jako że dodaliśmy już do systemu rodzaje, napiszemy  $\lambda X :: *. \lambda x : X.x : \forall X :: *. X \rightarrow X :: *$ .

Czy moglibyśmy napisać  $\lambda X :: * \Rightarrow *. \lambda x : X.x : \forall X :: * \Rightarrow *. X \rightarrow X :: * \Rightarrow *$ ? 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

$t ::=$	$\dots$	<i>termy</i>
	$\lambda X :: K.t$	<i>abstrakcja typowa</i>
	$t[T]$	<i>aplikacja typowa</i>

- typ uniwersalny

$T ::=$	$\dots$	<i>typy</i>
	$\forall X :: K.T$	<i>typ uniwersalny</i>

## 2.2. Typowanie

### 2.2.1. Kontekst

*Kontekst typowania* opisany jest następującą składnią abstrakcyjną:

$\Gamma ::=$		<i>kontekst</i>
	$\emptyset$	<i>pusty kontekst</i>
	$\Gamma, x : T$	<i>wiązanie typu</i>
	$\Gamma, X :: K$	<i>wiązanie rodzaju</i>

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

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

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.

- $[Y \mapsto T]X = \begin{cases} T & Y = X \\ X & \text{w.p.p} \end{cases}$
- $[Y \mapsto T](X_1 X_2) = [Y \mapsto T]X_1 [Y \mapsto T]X_2$
- $[Y \mapsto T](S_1 \rightarrow S_2) = [Y \mapsto T]S_1 \rightarrow [Y \mapsto T]S_2$
- $[Y \mapsto T]\forall X.S = \begin{cases} \forall X.S & Y = X \text{ lub } Y \notin FV(S) \\ \forall X.[Y \mapsto T]S & X \notin FV(S) \text{ i } Y \in FV(S) \end{cases}$
- $[Y := T]\lambda X.S = \begin{cases} \lambda X.S & Y = X \text{ lub } Y \notin FV(S) \\ \lambda X.[Y \mapsto T]S & X \notin FV(S) \text{ i } Y \in FV(S) \end{cases}$

### 2.2.3. Relacja $\equiv$

Jak wspomnieliśmy w rozdziale XXX, definiujemy na typach relację równoważności. W poniższych wzorach  $S, S_1, S_2, T, T_1, T_2$  to typy,  $K$  to rodzaj. Następujące trzy reguły:

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

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

$$\frac{S_1 \equiv T_1 \quad S_2 \equiv T_2}{S_1 \rightarrow S_2 \equiv T_1 \rightarrow T_2} \quad \frac{S_1 \equiv T_1 \quad S_2 \equiv T_2}{S_1 S_2 \equiv T_1 T_2}$$

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

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

#### 2.2.4. Reguły znajdowania rodzaju

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

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

$$\frac{X :: K \in \Gamma}{\Gamma \vdash X :: K} \quad \frac{\Gamma \vdash T_1 :: K_1 \Rightarrow K_2 \quad \Gamma \vdash T_2 :: K_1}{\Gamma \vdash T_1 T_2 :: K_2}$$

$$\frac{\Gamma \vdash X :: K_1 \quad \Gamma \vdash T :: K_2}{\Gamma \vdash \lambda X :: K_1.T :: K_1 \Rightarrow K_2} \quad \frac{\Gamma \vdash X :: K \quad \Gamma \vdash T :: *}{\Gamma \vdash \forall X :: K.T :: *}$$

$$\frac{\Gamma \vdash T_1 :: * \quad \Gamma \vdash T_2 :: *}{\Gamma \vdash T_1 \rightarrow T_2 :: *}$$

#### 2.2.5. Reguły typowania

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

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \frac{\Gamma \vdash T_1 :: * \quad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.t_2 : T_1 \rightarrow T_2}$$

$$\frac{\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1}{\Gamma \vdash t_1 t_2 : T_2} \quad \frac{\Gamma \vdash t : S \quad S \equiv T \quad \Gamma \vdash T :: *}{\Gamma \vdash t : T}$$

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

### 2.3. Ewaluacja

Wartości w  $F_\omega$  zdefiniujemy dokładnie jak w  $\lambda_{\rightarrow}$ .

$v ::=$	<i>wartości</i>
$\lambda x : T.t$	<i>wartość abstrakcji</i>

Ewaluacja przebiega w sposób standardowy dla aplikacji i abstrakcji termów. Teraz, dla czytelności, przetoczmy te reguły ewaluacji ( $t_1, t'_1, t_2, t'_2, t$  to termy,  $v$  to wartość,  $x:T$  to zmienna  $x$  typu  $T$ ):

$$\frac{t_1 \longrightarrow t'_1}{t_1 t_2 \longrightarrow t'_1 t_2} \quad \frac{t_2 \longrightarrow t'_2}{v_1 t_2 \longrightarrow v_1 t'_2}$$

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

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

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

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

### 3. Rozszerzenia $F_\omega$

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

Typowanie:

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

#### 3.1. wyrażenia arytmetyczne i logiczne

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

Tworzenie rodzaju:

$$\frac{}{\Gamma \vdash \text{Bool} :: *} \quad \frac{}{\Gamma \vdash \text{Nat} :: *}$$

Typowanie:

$$\frac{}{\Gamma \vdash \text{true} : \text{Bool}} \quad \frac{}{\Gamma \vdash \text{false} : \text{Bool}} \quad \frac{\Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T \quad \Gamma \vdash T :: *}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T}$$

$$\frac{}{\Gamma \vdash \text{zero} : \text{Nat}} \quad \frac{\Gamma \vdash t : \text{Nat}}{\Gamma \vdash \text{succ } t : \text{Nat}} \quad \frac{\Gamma \vdash t : \text{Nat}}{\Gamma \vdash \text{pred } t : \text{Nat}} \quad \frac{\Gamma \vdash t : \text{Nat}}{\Gamma \vdash \text{iszero } t : \text{Bool}}$$

Ewaluacja:

aaa

### 3.2. unit i sekwencje

W zasadzie unita można nie wprowadzać:

$\text{unit} = \lambda X :: *. \lambda x : X. x$

$\text{Unit} = \forall X :: *. X \rightarrow X$

Typowanie:

$$\frac{\Gamma \vdash \text{unit} : \text{Unit}}{\mathbf{t}_1; \mathbf{t}_2 = (\lambda x : \text{Unit}. \mathbf{t}_2) \mathbf{t}_1} \quad \text{gdzie } x \notin \text{FV}(t_2)$$

a to stare, może się przyda:

$\mathbf{t} ::=$	$\dots$	<i>termy</i>
	$\text{unit}$	
$\mathbf{T} ::=$	$\dots$	<i>typy</i>
	$\text{Unit}$	
$\mathbf{v} ::=$	$\dots$	<i>wartości</i>
	$\text{unit}$	

Tworzenie rodzaju:

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

Typowanie:

$$\frac{\Gamma \vdash \text{unit} : \text{Unit}}{\mathbf{t}_1; \mathbf{t}_2 = (\lambda x : \text{Unit}. \mathbf{t}_2) \mathbf{t}_1} \quad \text{gdzie } x \notin \text{FV}(t_2)$$

Ewaluacja:

aaa

### 3.3. Definicje lokalne

$$\text{let } x = \mathbf{t}_1 \text{ in } \mathbf{t}_2 \stackrel{\text{def}}{=} (\lambda x : \mathbf{T}. \mathbf{t}_2) \mathbf{t}_1 \quad \text{gdzie } \mathbf{t}_1 : \mathbf{T}$$

### 3.4. Rekordy

$\mathbf{t} ::=$	$\dots$	<i>termy</i>
	$\{l_i = \mathbf{t}_i^{i \in 1..n}\}$	<i>rekord</i>
	$\mathbf{t}.l$	<i>projekcja</i>
$\mathbf{T} ::=$	$\dots$	<i>typy</i>
	$\{l_i : \mathbf{T}_i^{i \in 1..n}\}$	<i>typ rekordu</i>
$\mathbf{v} ::=$	$\dots$	<i>wartości</i>
	$\{l_i = \mathbf{v}_i^{i \in 1..n}\}$	<i>wartość rekordu</i>

Tworzenie rodzaju:

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

Typowanie:

$$\frac{\Gamma \vdash t_1 : T_1 \dots \Gamma \vdash t_n : T_n \quad \Gamma \vdash \{l_i : T_i^{i \in 1..n}\} :: *}{\Gamma \vdash \{l_i = t_i^{i \in 1..n}\} : \{l_i : T_i^{i \in 1..n}\}} \quad \frac{\Gamma \vdash t : \{l_i : T_i^{i \in 1..n}\} \quad \Gamma \vdash \{l_i : T_i^{i \in 1..n}\} :: *}{\Gamma \vdash t.i : T_i}$$

Ewaluacja:

$$\frac{\{l_i = v_i^{i \in 1..n}\}.i \longrightarrow v_i \quad \frac{e \longrightarrow e'}{e.i \longrightarrow e'.i} \quad t_i \longrightarrow t'_i}{\{l_1 = v_1, \dots, l_{i-1} = v_{i-1}, l_i = t_i, \dots, l_n = t_n\} \longrightarrow \{l_1 = v_1, \dots, l_{i-1} = v_{i-1}, l_i = t'_i, \dots, l_n = t_n\}}$$

### 3.5. Warianty

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

Tworzenie rodzaju:

$$\frac{\Gamma \vdash T_1 :: * \dots \Gamma \vdash T_n :: *}{\Gamma \vdash \langle l_i : T_i^{i \in 1..n} \rangle :: *}$$

Typowanie:

$$\frac{\Gamma \vdash t_0 : \langle l_i : T_i^{i \in 1..n} \rangle \quad \Gamma, x_1 : T_1 \vdash t_1 : T \dots \Gamma, x_n : T_n \vdash t_n : T}{\Gamma \vdash \text{case } t_0 \text{ of } \langle l_i = x_i \Rightarrow t_i^{i \in 1..n} \rangle : T}$$

$$\frac{\Gamma \vdash t_j : T_j \quad \Gamma \vdash \langle l_i : T_i^{i \in 1..n} \rangle :: *}{\Gamma \vdash \langle l_j = t_j \rangle \text{ as } \langle l_i : T_i^{i \in 1..n} \rangle : \langle l_i : T_i^{i \in 1..n} \rangle}$$

Ewaluacja:

aaa

### 3.6. Punkt stały

$t ::=$	$\dots$	<i>termy</i>
	$\text{fix } t.v$	<i>punkt stały</i>

Typowanie

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

Ewaluacja

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



### 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 odwołają nas od tej konieczności.

$t ::=$	$\dots$	<i>termy</i>
	$\text{nil}[T]$	<i>lista pusta</i>
	$\text{cons}[T] \ t \ t$	<i>konstruktor listy</i>
	$\text{isnil}[T] \ t$	<i>test na pustość listy</i>
	$\text{head}[T] \ t \ t$	<i>głowa listy</i>
	$\text{tail}[T] \ t$	<i>ogon listy</i>
$T ::=$	$\dots$	<i>typy</i>
	$\text{List } T$	<i>typ listy</i>
$v ::=$	$\dots$	<i>typy</i>
	$\text{nil } [T]$	<i>wartość pustej listy</i>
	$\text{cons } [T] \ v \ v$	<i>wartość listy niepustej</i>

Tworzenie rodzaju:

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

Typowanie:

$$\frac{\Gamma \vdash \text{List } T :: *}{\Gamma \vdash \text{nil}[T] : \text{List } T} \quad \frac{\Gamma \vdash t_1 : T \quad \Gamma \vdash t_2 : \text{List } T}{\Gamma \vdash \text{List}[T] \ t_1 \ t_2 : \text{List } T}$$

$$\frac{\Gamma \vdash t : \text{List } T}{\Gamma \vdash \text{head}[T] \ t : T} \quad \frac{\Gamma \vdash t : \text{List } T}{\Gamma \vdash \text{tail}[T] \ t : \text{List } T}$$

Ewaluacja:

aaa

### 3.8. Typy egzystencjalne

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

$t ::=$	$\dots$	<i>termy</i>
	$\{ *T :: K, t \} \text{ as } T$	<i>pakowanie</i>
	$\text{let } \{ X, x \} = t \text{ in } t$	<i>odpakowanie</i>
$T ::=$	$\dots$	<i>typy</i>
	$\{ \exists X :: K, T \}$	<i>typ egzystencjalny</i>
$v ::=$	$\dots$	<i>wartości</i>
	$\{ *T, v \} \text{ as } T$	<i>pakowanie</i>

W systemie  $F_\omega$  powyższe elementy języka możemy zdefiniować następująco:

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

$$\begin{aligned} \{ *U :: K, t \} \text{ as } \{ \exists X :: K, T \} &\stackrel{\text{def}}{=} \text{let } x = t \text{ in } \lambda Y :: *. (\lambda f : \forall X :: K. T \rightarrow Y). f [U] x \\ \text{let } \{ X :: K, x \} = t \text{ in } t' &\stackrel{\text{def}}{=} t[T'](\lambda X :: K. \lambda x : T. t') \quad \text{gdzie } t' : T' \end{aligned}$$

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  $\{ *U :: K, t \} \text{ as } \{ \exists X :: K, T \}$ .

$$\begin{aligned} \{ *U :: K, t \} \text{ as } \{ \exists X :: K, T \} &= \text{let } x = t \text{ in } \lambda Y :: *. \lambda f : (\forall X :: K. T \rightarrow Y). f [U] x = \\ &= (\lambda x : [X \mapsto U]T. \lambda Y :: *. \lambda f : (\forall X :: K. T \rightarrow Y). f [U] x) t = \\ &\stackrel{t : [X \mapsto U]T}{=} \lambda Y :: *. \lambda f : (\forall X :: K. T \rightarrow Y). f [U] (t : [X \mapsto U]T) \end{aligned}$$

co jest typu  $\forall Y :: *. (\forall X :: K. T \rightarrow Y) \rightarrow Y$ , czyli z definicji  $\{ \exists X :: K, T \}$ .

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

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

Następnie wyprowadzimy ten term w  $F_\omega$ :

$$\begin{aligned} \{ \text{Nat} :: K, \{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} \} \text{ as } \{ \exists X :: K, \{ a : X, f : X \rightarrow \text{Nat} \} \} &= \\ = \text{let } x = \{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} \text{ in } \lambda Y :: *. \lambda f : (\forall X :: K. \{ a : X, f : X \rightarrow \text{Nat} \} \rightarrow Y). f [\text{Nat}] x &= \\ = (\lambda x : [X \mapsto \text{Nat}] \{ a : X, f : X \rightarrow \text{Nat} \}. \lambda Y :: *. \lambda f : (\forall X :: K. \{ a : X, f : X \rightarrow \text{Nat} \} \rightarrow Y). f [\text{Nat}] x) & \\ \{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} &\stackrel{\{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} : [X \mapsto \text{Nat}] \{ a : X, f : X \rightarrow \text{Nat} \}}{=} \\ = \lambda Y :: *. \lambda f : (\forall X :: K. \{ a : X, f : X \rightarrow \text{Nat} \} \rightarrow Y). f [\text{Nat}] (\{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} : [X \mapsto \text{Nat}] \{ a : X, f : X \rightarrow \text{Nat} \}) & \end{aligned}$$

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

Uważne odpakowanie otrzymanego termu pozostawiamy Czytelnikowi jako ćwiczenie, my pozwolimy sobie przeprowadzać schemat wywodu:

$$\begin{aligned} \text{let } \{ X, x \} &= \lambda Y :: *. \lambda f : (\forall X :: K. \{ a : X, f : X \rightarrow \text{Nat} \} \rightarrow Y). f [\text{Nat}] (\{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} : [X \mapsto \text{Nat}] \{ a : X, f : X \rightarrow \text{Nat} \}) \text{ in } (x.f \ x.a) = \\ &= (\lambda Y :: *. \lambda f : (\forall X :: K. \{ a : X, f : X \rightarrow \text{Nat} \} \rightarrow Y). f [\text{Nat}] (\{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} : [X \mapsto \text{Nat}] \{ a : X, f : X \rightarrow \text{Nat} \})) [\text{Nat}] (\{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} : [X \mapsto \text{Nat}] \{ a : X, f : X \rightarrow \text{Nat} \}) = \\ &= ((\lambda X :: K. \lambda x : T. (x.f \ x.a)) [\text{Nat}] (\{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \} : [X \mapsto \text{Nat}] \{ a : X, f : X \rightarrow \text{Nat} \})) = \\ &= (\{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \}. f \ \{ a = \text{zero}, f : \lambda x : \text{Nat}. \text{succ } x \}. a) = \\ &= (\lambda x : \text{Nat}. \text{succ } x) \text{zero} = \text{succ zero} \end{aligned}$$

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

Tworzenie rodzaju:

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

Typowanie:

$$\frac{\Gamma \vdash t : [X \mapsto U]T \quad \Gamma \vdash U :: K \quad \Gamma \vdash \{\exists X :: K, T\} :: *}{\Gamma \vdash \{ *U :: K, t \} \text{ as } \{\exists X :: K, T\} : \{\exists X :: K, T\}} \quad \frac{\Gamma \vdash t_1 : \{\exists X :: K, T_1\} \quad \Gamma, X :: K, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \text{let } \{X, x\} = t_1 \text{ in } t_2 : T_2}$$

Ewaluacja:

$$\begin{aligned} \text{let } \{X, x\} = (\{ *U :: K, v \} \text{ as } T) \text{ in } t &\longrightarrow [X \mapsto U][x \mapsto v]t \\ \frac{t &\longrightarrow t'}{\{ *U :: K, t \} \text{ as } T &\longrightarrow \{ *U :: K, t' \} \text{ as } T} \\ \frac{t_1 &\longrightarrow t'_1}{\text{let } \{X, x\} = t_1 \text{ in } t_2 &\longrightarrow \text{let } \{X, x\} = t'_1 \text{ in } t_2} \end{aligned}$$

### 3.9. Typy rekurencyjne

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

$t ::=$	...	<i>termy</i>
	$\text{fold}[T] \ t$	<i>folding</i>
	$\text{unfold}[T] \ t$	<i>unfolding</i>
$T ::=$	...	<i>typy</i>
	$\mu X.T$	<i>typ rekursywny</i>
$v ::=$	...	<i>wartości</i>
	$\text{fold}[T] \ v$	<i>folding</i>

Tworzenie rodzaju:

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

Typowanie:

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

Ewaluacja:

### 3.10. dopasowanie wzorca

## 4. Śladnia abstrakcyjna języka

## 5. Semantyka i typowanie

## 6. Rekonstrukcja typów

## 7. Własności i dowody

### 7.1. Inne własności $F_\omega$

**Definicja 1.** Reguły przepisywania typów w systemie  $F_\omega$  w wersji Curry'ego standardowe, oprócz:

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

Nierozstrzygalne są problemy:

- sprawdzania typu: dane  $\Gamma, M, \tau$ , 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,