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

Tematem naszej pracy jest system typów F_ω. Praca powstała jako część teoretyczna do projektu zaliczeniowego z systemu typów prowadzonego na Uniwersytecie Wrocławskim w roku akademickim 2010/2011. W pracy przedstawiliśmy ten system głównie od strony teoretycznej, zarówno minimalną część F_ω, jak i jego rozszerzenia, ale zaprezentowaliśmy również stworzony przez nas algorytm rekonstrukcji typów, choć rekonstrukcja typów w systemie F_ω jest nierozstrzygalna. Wady i własności tego algorytmu również przedstawiliśmy w pracy. Część praktyczna do projektu została napisana w języku F# i jest załączona do pracy.

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 rozdziale trzecim 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 anotowana*
	$\lambda\mathtt{x}.\mathtt{t}$	*abstrakcja nieanotowana*
	$\mathtt{t}\ \mathtt{t}$	*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 `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`.

Niestety, w tak zdefiniowanym systemie powstaje jeden problem. Nie chcielibyśmy, aby `Bool Bool` było dozwolone, tak samo, jak w świecie termów nie chcieliśmy, 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 $\lambda_{\to}$, są rodzaju $*$. Np. `Bool` :: $*$, `Nat` $\to$ `Nat`, (`Bool` $\to$ `Nat`) $\to$ `Nat` :: $*$, itd. Rodzaj $* \Rightarrow *$ będzie odpowiadał funkcjom z konstruktorów w konstruktory, np. λ`X`.`X` $\to$ `Bool` :: $* \Rightarrow *$. $* \Rightarrow * \Rightarrow *$ bierze konstruktor i zwraca funkcję konstruktorową, np. λ`X`.λ`Y`.`X` $\to$ `Y` :: $* \Rightarrow * \Rightarrow *$, itd.

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

- rodzaje

`K` ::=		*rodzaje*
	$*$	*rodzaj wszystkich typów*
	`K` $\Rightarrow$ `K`	*rodzaj funkcji typowej*

- abstrakcję i aplikację typową na typach

`T` ::=	...	*typy*
	λ`X` :: `K`.`T`	*abstrakcja konstruktorowa anotowana*
	λ`X`.`T`	*abstrakcja konstruktorowa nieanotowana*
	`T 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 λ`x`.`x`, w $\lambda_{\to}$ możemy ją napisać na wiele sposób: λ`x` : `Bool`.`x`, λ`x` : `Nat`.`x`, λ`x` : `Bool` $\to$ `Nat`.`x`. W systemie F możemy wszystkie te funkcje zapisać jako: λ`X`.λ`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: (λ`X`.λ`x` : `X`.`x`) [`Bool`] `true`, co daje `true`, albo (λ`X`.λ`x` : `X`.`x`) [`Nat`] `1`, co daje `1`. W ten sposób powstała nam *uniwersalna* funkcja identycznościowa, której nadamy tzw. uniwersalny typ: λ`X`.λ`x` : `X`.`x` : $\forall$`X`.`X` $\to$ `X`. Dodatkowo, jako że dodaliśmy już do systemu rodzaje, napiszemy λ`X` :: $*$.λ`x` : `X`.`x` : $\forall$`X` :: $*$.`X` $\to$ `X` :: $*$.

Czy moglibyśmy napisać λ`X` :: $* \Rightarrow *$.λ`x` : `X`.`x` : $\forall$`X` :: $* \Rightarrow *$.`X` $\to$ `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:

[1] formalnie zdefiniujemy tą relację w rozdziale 2.2.3

- abstrakcję i aplikację typową na termach

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

- typ uniwersalny

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

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 2.1.1, 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:

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

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

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

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

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

W rozdziale tym chcielibyśmy poruszyć, jak w systemie F_ω zdefiniować najprostsze konstrukcje, takie jak wyrażenia arytmetyczne i logiczne, warianty, sekwencje wyrażeń, typy egzystencjalne, rekordy i inne. Pokażemy również, jak przebiega typowanie, ewaluacja i gdzieniegdzie dodamy reguły tworzenia rodzaju.

Chcielibysmy podkreślić, że następująca reguła typowania:

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

bardzo ułatwia definiowanie reguł typowania w F_ω. W większości przypadków są one takie same lub lekko zmodyfikowane, dlatego nie powinny nastręczać trudności.

3.1. wyrażenia arytmetyczne i logiczne

Wyreżania arytmetyczne i logiczne to część, bez której żaden język się nie obędzie. Oczywiście można je sobie zakodować w systemie F_ω, ale normą są wbudowane w język wyrażenia. Termy, typy i wartości wyrażeń zdefiniujemy następująco:

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*

Na pewno musimy dodać reguły tworzenia rodzaju dla `Nat` i `Bool`, którym nadamy rodzaj $*$:

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

Typowanie wygląda dokładnie tak samo jak w rachunku lambda z typami prostymi. Możemy sobie pozwolić na takie reguły dzięki regule XXX. Przykładowo, nie tylko termy typu `Nat` mogą się dobrze otypować, gdy zaaplikujemy je do `succ`. Dla $\texttt{t} : (\lambda \texttt{X}.\texttt{X})\texttt{Nat}$ otrzymamy:

$$\frac{\dfrac{\Gamma \vdash \texttt{t} : (\lambda \texttt{X}.\texttt{X})\texttt{Nat} \qquad \Gamma \vdash (\lambda \texttt{X}.\texttt{X})\texttt{Nat} \equiv \texttt{Nat} \qquad \Gamma \vdash (\lambda \texttt{X}.\texttt{X})\texttt{Nat} :: *}{\Gamma \vdash \texttt{t} : \texttt{Nat}}}{\Gamma \vdash \texttt{succ t} : \texttt{Nat}}$$

Nie musimy również pisać reguł typu:

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

ponieważ posiadanie typu przez term $\texttt{t}_2$ gwarantuje nam, że ten typ będzie rodzaju $*$. Stąd, reguły typowania wyrażeń arytmetycznych i logicznych wyglądają następująco w F_ω:

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

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

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

Zdefiniowanie reguł ewaluacji pozostawiamy Czytelnikowi.

3.2. Unit i sekwencje

W rachunku lambda z typami prostymi dodaliśmy do składnię języka rozszerzaliśmy o konstrukcje takie, jak:

t ::=	...	*termy*
	`unit`	*term unit*
T ::=	...	*typy*
	`Unit`	*typ unit*
v ::=	...	*wartości*
	`unit`	*wartość unit*

natomiast typowanie przebiegało następująco:

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

a sekwencje definiowaliśmy jako:

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

Aby pozostać przy wbudowanym unit w język wystarczy dodać regułę znajdowania rodzaju dla typu `Unit`:

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

W rachunku F_ω pojawia się możliwość zakodowania `unit` i `Unit`. Robimy to w taki sposób:
$\texttt{unit} \stackrel{\texttt{def}}{=} = \lambda \texttt{X} :: *.\lambda \texttt{x} : \texttt{X}.\texttt{x}$
$\texttt{Unit} \stackrel{\texttt{def}}{=} \forall \texttt{X} :: *.\texttt{X} \to \texttt{X}$

3.3. Anotacje typowe

Anotacje typowe są przydatną konstrukcją używaną na przykład przy typach egzystencjalnych.

t ::=	···	*termy*
	t as T	*anotacja typowa*

Ewaluacja i typowanie nie zmieniają się.

3.4. Definicje lokalne

Do zbioru termów dodamy następujące dwie konstrukcje:

t ::=		
	let $x = t$ in t	*zmienna lokalna*
	tlet $X = T$ in t	*typ lokalny*

Oczywiście możemy zdefiniować `let` i `tlet` jako:

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

$$\texttt{tlet X} = \texttt{T in t} \stackrel{\texttt{def}}{=} (\lambda \texttt{X} :: \texttt{K}.\texttt{t})\texttt{T} \qquad \text{gdzie } \texttt{T} :: \texttt{K}$$

Reguły typowania i ewaluacji pozostają bez zmian dla `let`, ale zdefiniujemy je dla `tlet`. Typowanie:

$$\frac{\Gamma, \mathtt{X} : \mathtt{S} \vdash \mathtt{t} : \mathtt{T}}{\Gamma \vdash \mathtt{tlet\ X} = \mathtt{S\ in\ t} : [\mathtt{X} \mapsto \mathtt{S}]\mathtt{T}}$$

i ewaluacja:

$$\mathtt{tlet\ X} = \mathtt{S\ in\ t} \longrightarrow [\mathtt{X} \mapsto \mathtt{S}]\mathtt{t}$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{tlet\ X} = \mathtt{T\ in\ t} \longrightarrow \mathtt{tlet\ X} = \mathtt{T\ in\ t}'}$$

3.5. Pary

Składnię par zdefiniujemy następująco:

$\mathtt{t}$ ::=	...	*termy*
	$(\mathtt{t}, \mathtt{t})$	*para*
	$\mathtt{t.1}$	*projekcja na pierwszy element*
	$\mathtt{t.2}$	*projekcja na drugi element*
$\mathtt{T}$ ::=	...	*typy*
	$\mathtt{T}_1 \times \mathtt{T}_2$	*typ pary*
$\mathtt{v}$ ::=	...	*wartości*
	$(\mathtt{v}, \mathtt{v})$	*wartość pary*

Do relacji tworzenia rodzaju dodamy regułę nadającą rodzaj typowi $\{\mathtt{l_i} : \mathtt{T_i}^{\ \mathtt{i} \in 1..\mathtt{n}}\}$:

$$\frac{\Gamma \vdash \mathtt{T}_1 :: * \qquad \Gamma \vdash \mathtt{T}_2 :: *}{\Gamma \vdash \mathtt{T}_1 \times \mathtt{T}_2 :: *}$$

oraz wprowadzimy niewielkie zmiany w regułach typowania:

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2 \qquad \Gamma \vdash \mathtt{T}_1 \times \mathtt{T}_2 :: *}{\Gamma \vdash (\mathtt{t}_1, \mathtt{t}_2) : \mathtt{T}_1 \times \mathtt{T}_2}$$

$$\frac{\Gamma \vdash \mathtt{t} : \mathtt{T}_1 \times \mathtt{T}_2}{\Gamma \vdash \mathtt{t.1} : \mathtt{T}_1} \qquad \frac{\Gamma \vdash \mathtt{t} : \mathtt{T}_1 \times \mathtt{T}_2}{\Gamma \vdash \mathtt{t.2} : \mathtt{T}_2}$$

a ewaluację pozostawimy bez zmian:

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{t.i} \longrightarrow \mathtt{t}'\mathtt{.i}} \qquad \text{gdzie } \mathtt{i} \in \{1, 2\}$$

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

$$(\mathtt{v}_1, \mathtt{v}_2).\mathtt{i} \longrightarrow \mathtt{v}_\mathtt{i} \qquad \text{gdzie } \mathtt{i} \in \{1, 2\}$$

3.6. Rekordy

Składnię rekordów zdefiniujemy następująco:

$\mathtt{t} ::=$	$\ldots$	*termy*
	$\{\mathtt{l_i} = \mathtt{t_i}^{\,\mathtt{i \in 1..n}}\}$	*rekord*
	$\mathtt{t.l}$	*projekcja*
$\mathtt{T} ::=$	$\ldots$	*typy*
	$\{\mathtt{l_i} : \mathtt{T_i}^{\,\mathtt{i \in 1..n}}\}$	*typ rekordu*
$\mathtt{v} ::=$	$\ldots$	*wartości*
	$\{\mathtt{l_i} = \mathtt{v_i}^{\,\mathtt{i \in 1..n}}\}$	*wartość rekordu*

Do relacji tworzenia rodzaju dodamy regułę nadającą rodzaj typowi $\{\mathtt{l_i} : \mathtt{T_i}^{\,\mathtt{i \in 1..n}}\}$:

$$\frac{\Gamma \vdash \mathtt{T_1} :: * \ \ldots\ \Gamma \vdash \mathtt{T_n} :: *}{\Gamma \vdash \{\mathtt{l_i} : \mathtt{T_i}^{\,\mathtt{i \in 1..n}}\} :: *}$$

oraz wprowadzimy niewielkie zmiany w regułach typowania:

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1}\ \ldots\ \Gamma \vdash \mathtt{t_n} : \mathtt{T_n} \qquad \Gamma \vdash \{\mathtt{l_i} : \mathtt{T_i}^{\,\mathtt{i \in 1..n}}\} :: *}{\Gamma \vdash \{\mathtt{l_i} = \mathtt{t_i}^{\,\mathtt{i \in 1..n}}\} : \{\mathtt{l_i} : \mathtt{T_i}^{\,\mathtt{i \in 1..n}}\}}$$

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

a ewaluację pozostawimy bez zmian:

$$\{\mathtt{l_i} = \mathtt{v_i}^{\,\mathtt{i \in 1..n}}\}.\mathtt{i} \longrightarrow \mathtt{v_i} \qquad \frac{\mathtt{t} \longrightarrow \mathtt{t'}}{\mathtt{t.i} \longrightarrow \mathtt{t'.i}}$$

$$\frac{\mathtt{t_i} \longrightarrow \mathtt{t'_i}}{\{\mathtt{l_1} = \mathtt{v_1}, \ldots, \mathtt{l_{i-1}} = \mathtt{v_{i-1}}, \mathtt{l_i} = \mathtt{t_i}, \ldots, \mathtt{l_n} = \mathtt{t_n}\} \longrightarrow \{\mathtt{l_1} = \mathtt{v_1}, \ldots, \mathtt{l_{i-1}} = \mathtt{v_{i-1}}, \mathtt{l_i} = \mathtt{t'_i}, \ldots, \mathtt{l_n} = \mathtt{t_n}\}}$$

3.7. Sumy

Składnię sum zdefiniujemy następująco:

$\mathtt{t} ::=$	$\ldots$	*termy*
	$\mathtt{inl\ t}$	*tagowanie lewe*
	$\mathtt{inr\ t}$	*tagowanie prawe*
	$\mathtt{case\ t\ of\ inl\ x} \Rightarrow \mathtt{t} \mid \mathtt{inr\ x} \Rightarrow \mathtt{t}$	*case*
$\mathtt{T} ::=$	$\ldots$	*typy*
	$\mathtt{T} + \mathtt{T}$	*typ sumy*

Typ sumy dostanie rodzaj $*$:

$$\frac{\Gamma \vdash \mathtt{T_1} :: * \qquad \Gamma \vdash \mathtt{T_2} :: *}{\Gamma \vdash \mathtt{T_1} + \mathtt{T_2} :: *}$$

a w regułach typowania wprowadzimy małe zmiany:

$$\frac{\Gamma \vdash \mathtt{t_0} : \mathtt{T_1} + \mathtt{T_2} \qquad \Gamma, \mathtt{x_1} : \mathtt{T_1} \vdash \mathtt{t_1} : \mathtt{T} \qquad \Gamma, \mathtt{x_2} : \mathtt{T_2} \vdash \mathtt{t_2} : \mathtt{T}}{\Gamma \vdash \mathtt{case\ t_0\ of\ inl\ x_1} \Rightarrow \mathtt{t_1} \mid \mathtt{inr\ x_2} \Rightarrow \mathtt{t_2} : \mathtt{T}}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma \vdash \mathtt{T}_1 + \mathtt{T}_2 :: *}{\Gamma \vdash \mathtt{inl}\ \mathtt{t}_1 : \mathtt{T}_1 + \mathtt{T}_2}$$

$$\frac{\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_2 \qquad \Gamma \vdash \mathtt{T}_1 + \mathtt{T}_2 :: *}{\Gamma \vdash \mathtt{inr}\ \mathtt{t}_2 : \mathtt{T}_1 + \mathtt{T}_2}$$

Ewaluację pozostawiamy bez zmian.

3.8. Warianty

Składnię wariantów zdefiniujemy następująco:

$\mathtt{t} ::=$	$\ldots$	*termy*
	$< \mathtt{l} = \mathtt{t} >\ \mathtt{as}\ \mathtt{T}$	*tagowanie*
	$\mathtt{case}\ \mathtt{t}\ \mathtt{of}\ < \mathtt{l}_i = \mathtt{x}_i > \Rightarrow \mathtt{t}_i\ ^{i \in 1..n}$	*case*
$\mathtt{T} ::=$	$\ldots$	*typy*
	$< \mathtt{l}_i : \mathtt{T}_i\ ^{i \in 1..n} >$	*typ wariantu*

Podobnie jak przy rekordach, typ wariantu dostanie rodzaj $*$:

$$\frac{\Gamma \vdash \mathtt{T}_1 :: * \ \ldots\ \Gamma \vdash \mathtt{T}_n :: *}{\Gamma \vdash < \mathtt{l}_i : \mathtt{T}_i\ ^{i \in 1..n} > :: *}$$

a w regułach typowania wprowadzimy małe zmiany:

$$\frac{\Gamma \vdash \mathtt{t}_0 :< \mathtt{l}_i : \mathtt{T}_i\ ^{i \in 1..n} > \qquad \Gamma, \mathtt{x}_1 : \mathtt{T}_1 \vdash \mathtt{t}_1 : \mathtt{T}\ \ldots\ \Gamma, \mathtt{x}_n : \mathtt{T}_n \vdash \mathtt{t}_n : \mathtt{T}}{\Gamma \vdash \mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ < \mathtt{l}_i = \mathtt{x}_i > \Rightarrow \mathtt{t}_i\ ^{i \in 1..n} : \mathtt{T}}$$

$$\frac{\Gamma \vdash \mathtt{t}_j : \mathtt{T}_j \qquad \Gamma \vdash < \mathtt{l}_i : \mathtt{T}_i\ ^{i \in 1..n} > :: *}{\Gamma \vdash < \mathtt{l}_j = \mathtt{t}_j > \mathtt{as} < \mathtt{l}_i : \mathtt{T}_i\ ^{i \in 1..n} > :< \mathtt{l}_i :\ \mathtt{T}_i\ ^{i \in 1..n} >}$$

natomiast ewaluacja pozostanie bez zmian:

$$\mathtt{case}\ (< \mathtt{l}_j = \mathtt{t}_j >\ \mathtt{as}\ \mathtt{T})\ \mathtt{of}\ < \mathtt{l}_i = \mathtt{x}_i > \Rightarrow \mathtt{t}_i\ ^{i \in 1..n} \longrightarrow [\mathtt{x}_j \mapsto \mathtt{v}_j]\mathtt{t}_j$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{case}\ \mathtt{t}\ \mathtt{of}\ < \mathtt{l}_i = \mathtt{x}_i > \Rightarrow \mathtt{t}_i\ ^{i \in 1..n} \longrightarrow \mathtt{case}\ \mathtt{t}'\ \mathtt{of}\ < \mathtt{l}_i = \mathtt{x}_i > \Rightarrow \mathtt{t}_i\ ^{i \in 1..n}}$$

$$\frac{\mathtt{t}_j \longrightarrow \mathtt{t}'_j}{< \mathtt{l}_j = \mathtt{t}_j >\ \mathtt{as}\ \mathtt{T} \longrightarrow < \mathtt{l}_j = \mathtt{t}'_j >\ \mathtt{as}\ \mathtt{T}}$$

3.9. Punkt stały

$\mathtt{t} ::=$	$\ldots$	*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

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

3.10. 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*

Do reguł rodzajowania dodamy regułę:

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

czyli $\mathtt{List}\ \mathtt{T}$ ma ten sam rodzaj, co jego elementy. Reguły typowania nieznacznie się zmienią:

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

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

a ewaluacja pozostanie ta sama, uzupełnienie jej pozostawiamy Czytelnikowi.

3.11. 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:

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

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

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

Pokażemy, że zachodzą podstawowe własności pakowania i odpakowania.
Rozważmy term $\{^*\mathtt{U} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as}\ \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$.
$\{^*\mathtt{U} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as}\ \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$
$= \mathtt{let}\ \mathtt{x} = \mathtt{t}\ \mathtt{in}\ \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}]\mathtt{x} =$
$= (\lambda \mathtt{x} : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{T}.\lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}]\mathtt{x})\mathtt{t} =$
$\overset{\mathtt{t}:[\mathtt{X}\mapsto\mathtt{U}]\mathtt{T}}{=} \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}](\mathtt{t} : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{T})$
co jest typu $\forall \mathtt{Y} :: *.(\forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}) \to \mathtt{Y}$, 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:

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

Następnie wyprowadzimy ten term w F_ω:

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

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:
$\mathtt{let}\ \{\mathtt{X}, \mathtt{x}\} = \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\} \to \mathtt{Y}).\mathtt{f}[\mathtt{Nat}]\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat}.\mathtt{succ}\ \mathtt{x}\}\ \mathtt{in}\ (\mathtt{x}.\mathtt{f}\ \mathtt{x}.\mathtt{a}) =$
$= (\lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\} \to \mathtt{Y}).\mathtt{f}[\mathtt{Nat}]\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat}.\mathtt{succ}\ \mathtt{x}\})[\mathtt{T}'](\lambda \mathtt{X} :: \mathtt{K}.\lambda \mathtt{x} : \mathtt{T}.(\mathtt{x}.\mathtt{f}\ \mathtt{x}.\mathtt{a})) =$
$= ((\lambda \mathtt{X} :: \mathtt{K}.\lambda \mathtt{x} : \mathtt{T}.(\mathtt{x}.\mathtt{f}\ \mathtt{x}.\mathtt{a}))[\mathtt{Nat}](\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat}.\mathtt{succ}\ \mathtt{x}\} : [\mathtt{X} \mapsto \mathtt{Nat}]\{\mathtt{a} : \mathtt{X}, \mathtt{f} : \mathtt{X} \to \mathtt{Nat}\})) =$
$= (\{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat}.\mathtt{succ}\ \mathtt{x}\}.\mathtt{f}\ \{\mathtt{a} = \mathtt{zero}, \mathtt{f} : \lambda \mathtt{x} : \mathtt{Nat}.\mathtt{succ}\ \mathtt{x}\}.\mathtt{a}) =$
$= (\lambda \mathtt{x} : \mathtt{Nat}.\mathtt{succ}\ \mathtt{x})\mathtt{zero} = \mathtt{succ}\ \mathtt{zero}$

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.

Na pewno musimy określić, jaki rodzaj dostaje $\{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$:

$$\frac{\Gamma, \mathtt{X} :: \mathtt{K} \vdash \mathtt{T} :: *}{\Gamma \vdash \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\} :: *}$$

oraz zdefiniować relację znajdowania typu odpowiednią dla F_ω:

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

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

Ewaluacja nieznacznie się zmieni:

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

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

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}'_1}{\mathtt{let}\ \{\mathtt{X}, \mathtt{x}\} = \mathtt{t}_1\ \ \mathtt{in}\ \mathtt{t}_2 \longrightarrow \mathtt{let}\ \{\mathtt{X}, \mathtt{x}\} = \mathtt{t}'_1\ \ \mathtt{in}\ \mathtt{t}_2}$$

3.12. Uzupełnianie relacji $\equiv$

Po wprowadzeniu rozszerzeń z rozdziału trzeciego możemy rozszerzyć relację $\equiv$:

$$\frac{\mathtt{S} \equiv \mathtt{T}}{\{\exists \mathtt{X} :: \mathtt{K}, \mathtt{S}\} \equiv \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}} \qquad \frac{\mathtt{S} \equiv \mathtt{T}}{\mathtt{List}\ \mathtt{S} \equiv \mathtt{List}\ \mathtt{T}}$$

$$\frac{\mathtt{T}_1 \equiv \mathtt{T}'_1 \ldots \mathtt{T}_n \equiv \mathtt{T}'_n}{\{\mathtt{l}_i : \mathtt{T}_i\ ^{i \in 1..n}\} \equiv \{\mathtt{l}_i : \mathtt{T}'_i\ ^{i \in 1..n}\}} \qquad \frac{\mathtt{T}_1 \equiv \mathtt{T}'_1 \ldots \mathtt{T}_n \equiv \mathtt{T}'_n}{< \mathtt{l}_i : \mathtt{T}_i\ ^{i \in 1..n} > \equiv < \mathtt{l}_i : \mathtt{T}'_i\ ^{i \in 1..n} >}$$

4. Rekonstrukcja typów

4.1. Kilka słów o składni.

Rozważmy język z rozdziału drugiego rozszerzony o następujące konstrukcje:

t ::=		
	let $x = t$ in t	*zmienna lokalna*
	tlet $X = T$ in t	*typ lokalny*
T ::=		
	$\bar{X}$	*zmienna typowa (kwantyfikowana schematem)*
K ::=		
	$\widehat{X}$	*zmienna rodzajowa (kwantyfikowana schematem)*

W dodatku zakładamy, że wszystkie konstruktorowe zmienne związane są unikatowe (tzn. zmienna X może występować za conajwyżej jednym kwantyfikatorem).

Ważnym założeniem jest to, że $\bar{X} :: *$, co nam ułatwi sprawę przy unifikacji, a zarazem zbytnio nie ograniczy języka.

Jeżeli chcemy dodać ML-polimorfizm, będą nam portrzebne schematy typów i schematy rodzajów:

$\langle T \rangle$ ::=		*schemat typu*
	T	*typ schematu*
	$\Omega\bar{X}.\langle T \rangle$	*kwantyfikacja zmiennej typowej*
	$\Omega\widehat{X}.\langle T \rangle$	*kwantyfikacja zmiennej rodzajowej*
$\langle K \rangle$ ::=		*schemat rodzaju*
	K	*rodzaj schematu*
	$\Omega\widehat{X}.\langle K \rangle$	*kwantyfikacja zmiennej rodzajowej*

Będziemy używać następującego cukru syntaktycznego:

$$\Omega\bar{X}_1 \ldots \bar{X}_n\widehat{X}_1 \ldots \widehat{X}_m.T \equiv \Omega\bar{X}_1.\ldots\Omega\bar{X}_n.\Omega\widehat{X}_1.\ldots\Omega\widehat{X}_m.T$$

I analogicznie dla rodzajów.

Teraz sam kontekst typowania ma postać:

$$\begin{array}{l}\Gamma ::= \\ \qquad \emptyset \\ \qquad \Gamma, x : \langle T \rangle \\ \qquad \Gamma, X :: \langle K \rangle\end{array}$$

Jeżeli chcemy spojrzeć na ten język, jak na język programowania z rekonstrukcją typów, można dopuścić nieanotowane kwantyfikatory, które parser będzie zamieniać na anotowane świeżą zmienną schematową (typową albo rodzajową w zależności od kwantyfikatora). Z drugiej strony nie ma potrzeby pozwalać programiście jawnie korzystać ze zmiennych schematowych.

4.2. Zbiory zmiennych i podstawienia

W dalszej części będziemy używać różnych zbiorów zmiennych występujących w konstrukcjach w naszym języku, więc to jest dobre miejsce by te zbiory i operacje zdefiniować.

4.2.1. Wszystkie zmienne schematowe

Definicja 1. *Zbiorem schematowych zmiennych typowych typu/schematu T* nazwiemy zbiór $\bar{Var}(T)$ zdefiniowany następująco

$$\begin{aligned}\bar{Var}(X) &= \emptyset \\ \bar{Var}(T_1 \to T_2) &= \bar{Var}(T_1) \cup \bar{Var}(T_2) \\ \bar{Var}(\forall X :: K.T) &= \bar{Var}(T) \\ \bar{Var}(\lambda X :: K.T) &= \bar{Var}(T) \\ \bar{Var}(T_1\ T_2) &= \bar{Var}(T_1) \cup \bar{Var}(T_2) \\ \bar{Var}(\bar{X}) &= \{\bar{X}\}\end{aligned}$$

$$\begin{aligned}\bar{Var}(\Omega\bar{X}.\langle T \rangle) &= \{\bar{X}\} \cup \bar{Var}(\langle T \rangle) \\ \bar{Var}(\Omega\widehat{X}.\langle T \rangle) &= \bar{Var}(\langle T \rangle)\end{aligned}$$

Definicja 2. Anologicznie definijuemy *zbiór zmiennych rodzajowych rodzaju/typu/schematu T*.

$$\begin{aligned}\widehat{Var}(*) &= \emptyset \\ \widehat{Var}(K_1 \Rightarrow K_2) &= \widehat{Var}(K_1) \cup \widehat{Var}(K_2) \\ \widehat{Var}(\widehat{X}) &= \{\widehat{X}\}\end{aligned}$$

$$
\begin{aligned}
\widehat{Var}(X) &= \emptyset \\
\widehat{Var}(T_1 \to T_2) &= \widehat{Var}(T_1) \cup \widehat{Var}(T_2) \\
\widehat{Var}(\forall X :: K.T) &= \widehat{Var}(K) \cup \widehat{Var}(T) \\
\widehat{Var}(\lambda X :: K.T) &= \widehat{Var}(K) \cup \widehat{Var}(T) \\
\widehat{Var}(T_1\ T_2) &= \widehat{Var}(T_1) \cup \widehat{Var}(T_2) \\
\widehat{Var}(\widehat{X}) &= \emptyset
\end{aligned}
$$

$$
\begin{aligned}
\widehat{Var}(\Omega \bar{X}.\, \langle T \rangle) &= \widehat{Var}(\langle T \rangle) \\
\widehat{Var}(\Omega \widehat{X}.\, \langle T \rangle) &= \{\widehat{X}\} \cup \widehat{Var}(\langle T \rangle)
\end{aligned}
$$

4.2.2. Zmienne typowe wolne i związane

Definicja 3. *Zbiór zmiennych typowych wolnych termu/typu T* nazwiemy zbiór $FTV(T)$ zdefiniowany następująco

$$
\begin{aligned}
FTV(X) &= X \\
FTV(T_1 \to T_2) &= FTV(T_1) \cup FTV(T_2) \\
FTV(\forall X :: K.T) &= FTV(T) \setminus \{X\} \\
FTV(\lambda X :: K.T) &= FTV(T) \setminus \{X\} \\
FTV(T_1\ T_2) &= FTV(T_1) \cup FTV(T_2) \\
FTV(\bar{X}) &= \emptyset
\end{aligned}
$$

$$
\begin{aligned}
FTV(x) &= \emptyset \\
FTV(\lambda x : T.t) &= FTV(T) \cup FTV(t) \\
FTV(t_1\ t_2) &= FTV(t_1) \cup FTV(t_2) \\
FTV(\lambda X :: K.t) &= FTV(t) \setminus \{X\} \\
FTV(t[T]) &= FTV(t) \cup FTV(T) \\
FTV(\text{let } x = t_1 \text{ in } t_2) &= FTV(t_1) \cup FTV(t_2) \\
FTV(\text{tlet } X = T \text{ in } t) &= FTV(T) \cup (FTV(t) \setminus \{X\})
\end{aligned}
$$

Definicja 4. *Zbiór zmiennych typowych związanych termu/typu T* nazwiemy zbiór $BTV(T)$ zdefiniowany następująco

$$
\begin{aligned}
BTV(X) &= \emptyset \\
BTV(T_1 \to T_2) &= BTV(T_1) \cup BTV(T_2) \\
BTV(\forall X :: K.T) &= BTV(T) \cup \{X\} \\
BTV(\lambda X :: K.T) &= BTV(T) \cup \{X\} \\
BTV(T_1\ T_2) &= BTV(T_1) \cup BTV(T_2) \\
BTV(\bar{X}) &= \emptyset
\end{aligned}
$$

$$
\begin{aligned}
BTV(x) &= \emptyset \\
BTV(\lambda x : T.t) &= BTV(T) \cup BTV(t) \\
BTV(t_1\ t_2) &= BTV(t_1) \cup BTV(t_2) \\
BTV(\lambda X :: K.t) &= BTV(t) \cup \{X\} \\
BTV(t[T]) &= BTV(t) \cup BTV(T) \\
BTV(\text{let } x = t_1 \text{ in } t_2) &= BTV(t_1) \cup BTV(t_2) \\
BTV(\text{tlet } X = T \text{ in } t) &= BTV(T) \cup BTV(t) \cup \{X\}
\end{aligned}
$$

4.2.3. Podstawienia

Założenie o unikalności zmiennych związanych wiąże się z niewielką modyfkacją podstawienia.

$$
\{X := T\}Y = \begin{cases} \alpha T & X = Y, \alpha \text{ jest przemianowaniem zmiennych} \\ & \text{związanych w } T \text{ na świeże} \\ Y & X \neq Y \end{cases}
$$

Reszta definicji pozostaje bez zmian.

Przy rekonstrukcji typów będziemy dodatkowo używać podstawienia, które dodatkowo wstawia świeże zmienne rodzajowe. Zdefiniujemy je podobnie:

$$
\{X(\{\widehat{X}_1, \ldots, \widehat{X}_n\}) := T\}Y = \begin{cases} \alpha\sigma T & X = Y, \alpha \text{ jest przemianowaniem zmiennych} \\ & \text{związanych w } T \text{ na świeże,} \\ & \sigma = [\widehat{X}_1 := fresh, \ldots, \widehat{X}_n := fresh] \\ Y & X \neq Y \\ & \ldots \end{cases}
$$

σ jest podstawieniem schematowym, definicja jego działania pojawi się później.

Purystów matematycznych może przerażać pojawiająca się w definicji „świeża zmienna", oznaczana jako $fresh$. Jest to zmienna która dotychczas nigdzie się nie pojawiła, i nie będzie wprowadzona więcej niż raz. Można ją formalnie zdefiniować pamiętając cały czas zbiór użytych zmiennych i przekazując go jako dodatkowy argument do prawie wszystkich operacji, ale spowodowało by to znaczne pogorszenie czytelności tego dokumentu, dlatego tak nie robimy. Zainteresowany Czytelnik może samodzielnie poprawić przytoczone tu definicje i twierdzenia formalnie definiując świeżą zmienną.

Dla przytoczonych tu definicji później zajdzie potrzeba rozszerzenia ich na konteksty i podstawienia. Rozszerzenia te są na tyle oczywiste i naturalne, że nie ma potrzeby ich przytaczania.

4.3. β-unifikacja

Podczas rekonstukcji typów pojawiają się równania więzów które należy rozwiązać. Samo rozwiązanie sprowadza się do unifikacji pewnych termów, ale tutaj, ze względu na możliwość występowania funkcji typowych, termy równoważne nie muszą być równe, więc sama unifikacja powinna sprowadzać termy do β-równych sobie.

Taka unifikacja niesie ze sobą wiele problemów. Po pierwsze, będziemy chcieli używać podstawienia (za zmienne schematowe), będącego unifikatorem, na innych termach. To zaś grozi uzewnętrznieniem zmiennych związanych. Drugi problem to taki, że samo podstawienie nie musi zrównywać termów, które da się zrównać, ale różnią się nazwami zmiennych związanych.

Z pierwszym problemem poradzimy sobie, traktując podstawienia jako funkcję częściową, tzn. samo podstawienie oprócz przyporządkowań postaci $[\bar{X} := T]$, może również zawierać podstawienia postaci $[\bar{X} := fail]$. Dodając operację anulowania podstawienia otrzymujemy:

$$[\bar{X} := T] \setminus X = \begin{cases} [\bar{X} := fail] & X \in FTV(T) \\ [\bar{X} := T] & X \notin FTV(T) \end{cases}$$

Z drugim problemem poradzimy sobie traktując unifikator jako parę zawierającą podstawienie, które dobrze działa na zewnątrz, oraz term będący wynikiem unifikacji.

Definicja 5. *Podstawieniem schematowym* nazwiemy skończony zbiór par postaci $(\bar{X}, T)$, $(\bar{X}, fail)$ oraz $(\widehat{X}, K)$, takich że $\bar{X} \notin \bar{Var}(T)$ oraz $\widehat{X} \notin \widehat{Var}(K)$ (po podstawieniu powinny zniknąć wszystkie zmienne, za które coś podstawiliśmy). *Podstawieniem schematowym pustym* nazwiemy podstawienie schematowe będące zbiorem pustym i będziemy oznaczać przez []. Podstawienia schematowe będziemy reprezentować jako listy postaci

$$[\bar{X}_1 := T_1, \ldots, \bar{X}_k := T_k, \bar{X}_{k+1} := fail, \ldots, \bar{X}_n := fail, \widehat{X}_1 := K_1, \ldots, \widehat{X}_m := K_m].$$

Dziedziną podstawienia schematowego σ nazwiemy zbiór takich zmiennych typowych $\bar{X}$ i rodzajowych $\widehat{X}$, że $(\bar{X}, T) \in \sigma$ lub $(\bar{X}, fail) \in \sigma$ oraz $(\widehat{X}, K) \in \sigma$. Dziedzinę podstawienia schematowego σ będziemy oznaczać przez $Dom(\sigma)$.

Definicja 6. Podstawienie schematowe może działać na rodzajach, typach, termach i kontekstach. Jeżeli w podstawieniu występuje para $\widehat{X} := K$, to za wszystkie wystąpienia zmiennej $\widehat{X}$ zostanie podstawione K. Analogicznie się dzieje w przypadku wystąpienia pary $\bar{X} := T$. Jeżeli w podstawieniu schematowym wystąpi para $\bar{X} := fail$, a w typie/termie/kontekście do którego podstawienie aplikujemy występuje zmienna $\bar{X}$, to podstawienie jest niemożliwe.

Niech σ będzie podstawieniem schematowym. Działanie podstawienia formalnie definiujemy jako:

$$\sigma * = *$$
$$\sigma(K_1 \Rightarrow K_2) = \sigma K_1 \Rightarrow \sigma K_2$$
$$\sigma \widehat{X} = \begin{cases} K & (\widehat{X}, K) \in \sigma \\ \widehat{X} & \text{wpp} \end{cases}$$

$$
\begin{aligned}
\sigma X &= X \\
\sigma(T_1 \to T_2) &= \sigma T_1 \to \sigma T_2 \\
\sigma(\forall X :: K.T) &= \forall X :: \sigma K.\sigma T \\
\sigma(\lambda X :: K.T) &= \lambda X :: \sigma K.\sigma T \\
\sigma(T_1\ T_2) &= (\sigma T_1)\ (\sigma T_2) \\
\sigma\bar{X} &= \begin{cases} \alpha T & (\bar{X}, T) \in \sigma, \alpha \text{ jest przemianowaniem} \\ & \text{wprowadzającym świeże zmienne związane} \\ fail & (\bar{X}, fail) \in \sigma \\ \bar{X} & \text{wpp} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\sigma x &= x \\
\sigma(\lambda x : T.t) &= \lambda x : \sigma T.\sigma t \\
\sigma(t_1\ t_2) &= (\sigma t_1)\ (\sigma t_2) \\
\sigma(\lambda X :: K.t) &= \lambda X :: \sigma X.\sigma t \\
\sigma(t[T]) &= (\sigma t)[\sigma T] \\
\sigma(\text{let } x = t_1 \text{ in } t_2) &= \text{let } x = \sigma t_1 \text{ in } \sigma t_2 \\
\sigma(\text{tlet } x = T \text{ in } t) &= \text{tlet } x = \sigma T \text{ in } \sigma t
\end{aligned}
$$

$$
\sigma(\Omega\bar{X}.\langle T\rangle) = \begin{cases} \Omega\bar{X}.\sigma\langle T\rangle & \bar{X} \notin Dom(\sigma) \\ \Omega\bar{Y}.\sigma[\bar{X} := \bar{Y}]\langle T\rangle & \bar{X} \in Dom(\sigma) \wedge \bar{Y} \notin Dom(\sigma) \wedge \bar{Y} \notin \bar{Var}(\langle T\rangle) \end{cases}
$$

$$
\sigma(\Omega\widehat{X}.\langle T\rangle) = \begin{cases} \Omega\widehat{X}.\sigma\langle T\rangle & \widehat{X} \notin Dom(\sigma) \\ \Omega\widehat{Y}.\sigma[\widehat{X} := \widehat{Y}]\langle T\rangle & \widehat{X} \in Dom(\sigma) \wedge \widehat{Y} \notin Dom(\sigma) \wedge \widehat{Y} \notin \widehat{Var}(\langle T\rangle) \end{cases}
$$

$$
\begin{aligned}
\sigma\emptyset &= \emptyset \\
\sigma(\Gamma, x : \langle T\rangle) &= \sigma\Gamma, x : \sigma\langle T\rangle \\
\sigma(\Gamma, X :: \langle K\rangle) &= \sigma\Gamma, X :: \sigma\langle K\rangle
\end{aligned}
$$

Definicja 7. Powiemy, że podstawienie schematowe ρ jest *złożeniem* podstawień schematowych σ i θ, które będziemy oznaczać $\sigma \circ \theta$ jeżeli dla każdego rodzaju K, typu T, termu t i kontekstu Γ zachodzi:

$$
\begin{aligned}
\rho K &= \sigma(\theta K) \\
\rho T &= \sigma(\theta T) \\
\rho t &= \sigma(\theta t) \\
\rho\Gamma &= \sigma(\theta\Gamma)
\end{aligned}
$$

Lemat 1. *Niech σ i θ będą podstawieniami schematowymi. Wówczas zachodzi*

$$\begin{aligned}\sigma \circ \theta =& \{(\widehat{X}, \sigma K) | (\widehat{X}, K) \in \theta\} \\ & \cup \{(\widehat{X}, K) \in \sigma | \widehat{X} \notin Dom(\theta)\} \\ & \cup \{(\bar{X}, \sigma T) | (\bar{X}, T) \in \theta \wedge \sigma T \neq fail\} \\ & \cup \{(\bar{X}, fail) | \exists (\bar{X}, T) \in \theta . \sigma T = fail\} \\ & \cup \{(\bar{X}, T) \in \sigma | \bar{X} \notin Dom(\theta)\} \\ & \cup \{(\bar{X}, fail) \in \sigma | \bar{X} \notin Dom(\theta)\}\end{aligned}$$

Pozwala nam to algorytmicznie wyliczać złożenia podstawień schematowych.

Definicja 8. *Anulowaniem zbioru A zmiennych konstruktorowych z podstawienia schematowego σ* nazwiemy operację zdefiniowaną następująco:

$$\begin{aligned}\sigma \setminus A =& \{(\widehat{X}, K) \in \sigma\} \cup \{(\bar{X}, fail) \in \sigma\} \\ & \cup \{(\bar{X}, T) \in \sigma | FTV(T) \cap A = \emptyset\} \\ & \cup \{(\bar{X}, fail) | \exists (\bar{X}, T) \in \sigma . FTV(T) \cap A \neq \emptyset\}\end{aligned}$$

Anulowanie zmiennej X z podstawienia schematowego σ oznaczamy przez $\sigma \setminus X$ i definiujemy jako $\sigma \setminus \{X\}$.

Definicja 9. *β-unifikatorem* dla konstruktorów typów T_1 i T_2 nazwiemy taką parę (σ, S), że istnieje podstawienie schematowe θ i przemianowania zmiennych związanych α_1 i α_2 takie, że:

$$\theta\alpha_1 T_1 =_\beta S =_\beta \theta\alpha_2 T_2$$

oraz

$$\theta \setminus BTV(S) = \sigma$$

Definicja 10. Powiemy, że β-unifikator (σ, S) jest *ogólniejszy* od β-unifikatora (θ, T), jeżeli istnieje takie podstawienie schematowe ρ i przemianowanie α, że

$$\rho\alpha S =_\beta T \quad \text{oraz} \quad \rho\alpha\sigma = \theta$$

Definicja 11. *Najogólniejszym β-unifikatorem* dla konstruktorów typów T_1 i T_2 nazwiemy taki β-unifikator, który jest ogólniejszy od wszystkich innych β-unifikatorów tychże konstruktorów typów.

Lemat 2. *Niech A i B będą podzbiorami zbioru zmiennych konstruktorowych, niech θ będzie podstawieniem schematowym. Wówczas zachodzi*

$$(\theta \setminus A) \setminus B = \theta \setminus (A \cup B).$$

Lemat 3. *Niech A będzie podzbiorem zbioru zmiennych konstruktorowych, niech θ i ρ będą podstawieniami schematowymi, oraz niech zachodzi $FTV(\rho) \cap A = \emptyset$. Wówczas zachodzi*

$$(\rho \circ \theta) \setminus A = \rho \circ (\theta \setminus A)$$

.

Lemat 4. *Niech A będzie podzbiorem zbioru zmiennych konstruktorowych, niech θ będzie dowolnym podstawieniem schematowym, niech ρ będzie podstawieniem schematowym takim, że $FTV(\rho) \cap A = \emptyset$ (w szczególności zawierającym tylko zmienne rodzajowe). Wówczas zachodzi*

$$(\theta \circ \rho) \setminus A = (\theta \setminus A) \circ \rho$$

Lemat 5. *Niech (σ, S) będzie β-unifikatorem dla konstruktorów typów T_1 i T_2, oraz niech ρ będzie podstawieniem schematowym takim, że $FTV(\rho) \cap BTV(S) = \emptyset$. Wówczas $(\rho \circ \sigma, \rho S)$ również jest β-unifikatorem dla konstruktorów typów T_1 i T_2.*

Dowód. Wiemy, że istnieją takie θ, α_1 i α_2, że

$$\theta\alpha_1 T_1 =_\beta S =_\beta \theta\alpha_2 T_2 \quad \text{oraz} \quad \theta \setminus BTV(S) = \sigma$$

A wtedy zachodzi

$$(\rho \circ \theta)\alpha_1 T_1 = \rho(\theta\alpha_1 T_1) =_\beta \rho S =_\beta \rho(\theta\alpha_2 T_2) = (\rho \circ \theta_2)\alpha_2 T_2$$

oraz z lematu 3

$$(\rho \circ \theta) \setminus BTV(S) = \rho \circ (\theta \setminus BTV(S)) = \rho \circ \sigma$$

□

Lemat 6. *Niech T_1 i T_2 będą konstruktorami typów, ρ niech będzie podstawieniem schematowym takim, że $FTV(\rho) \cap BTV(T_1) = \emptyset$ oraz $FTV(\rho) \cap BTV(T_2) = \emptyset$, a (σ, S) niech będzie β-unifikatorem konstruktorów $(\rho T_1, \rho T_2)$. Wówczas $(\sigma \circ \rho, S)$ jest β-unifikatorem konstruktorów (T_1, T_2).*

Dowód. Niech $i \in \{1, 2\}$.
Z tego, że (σ, S) jest β-unifikatorem konstruktorów $(\rho T_1, \rho T_2)$ wiemy, że istnieje takie θ oraz α_i, że

$$\theta\alpha_i \rho T_i =_\beta S \quad \text{oraz} \quad \theta \setminus BTV(S) = \sigma$$

A wtedy z tego, że $FTV(\rho) \cap BTV(T_i) = \emptyset$

$$(\theta \circ \rho)\alpha_i T_i = \theta\alpha_i \rho T_i =_\beta S$$

oraz z lematu 4 mamy

$$(\theta \circ \rho) \setminus BTV(S) = (\theta \circ BTV(S)) \circ \rho = \sigma \circ \rho$$

Co kończy dowód. □

4.3.1. Algorytm β-unifikacji

Przy unifikacji wszystkie nieanotowane kwantyfikatory traktujemy jak anotowane unikatową zmienną. Funkcja $unify$ to klasyczna unifikacja (na rodzajach).
$unify_\beta(T_1, T_2)$ when $T_1 =_\beta T_2$ =
$([], T_1)$

$unify_\beta(\bar{X}, T) =$
    if $\bar{X} \in \bar{Var}(T)$ then $fail$
    else $([\bar{X} := T], T)$
$unify_\beta(T, \bar{X}) =$
    if $\bar{X} \in \bar{Var}(T)$ then $fail$
    else $([\bar{X} := T], T)$
$unify_\beta(\forall X_1 :: K_1.T_1, \forall X_2 :: K_2.T_2) =$
    let $\sigma_K = unify(K_1, K_2)$ in
    let $X_3 = fresh$ in
    let $(\sigma_T, T) = unify_\beta(\sigma_K\sigma_K\{X_1 := X_3\}T_1, \sigma_K\{X_2 := X_3\}T_2)$ in
        $((\sigma_T \setminus X_3) \circ \sigma_K, \forall X_3 :: \sigma_T\sigma_K K_1.T)$
$unify_\beta(\lambda X_1 :: K_1.T_1, \lambda X_2 :: K_2.T_2) =$
    let $\sigma_K = unify(K_1, K_2)$ in
    let $X_3 = fresh$ in
    let $(\sigma_T, T) = unify_\beta(\sigma_K\{X_1 := X_3\}T_1, \sigma_K\{X_2 := X_3\}T_2)$ in
        $((\sigma_T \setminus X_3) \circ \sigma_K, \lambda X_3 :: \sigma_T\sigma_K K_1.T)$
$unify_\beta(T_1 \to S_1, T_2 \to S_2) =$
    let $(\theta, T) = unify_\beta(T_1, T_2)$ in
    let $(\sigma, S) = unify_\beta(\theta S_1, \theta S_2)$ in
        $(\sigma \circ \theta, \sigma T \to S)$
$unify_\beta(T_1, T_2)$ when $T_1 \longrightarrow_\beta S_1$ =
    $unify_\beta(S_1, T_2)$
$unify_\beta(T_1, T_2)$ when $T_2 \longrightarrow_\beta S_2$ =
    $unify_\beta(T_1, S_2)$
$unify_\beta(V_1\ T_1, V_2\ T_2) =$
    let $(\theta, V) = unify_\beta(V_1, V_2)$ in
    let $(\sigma, T) = unify_\beta(\theta T_1, \theta T_2)$ in
        $(\sigma \circ \theta, \sigma V\ T)$

4.3.2. Własności

Fakt 1. *Algorytm $unify_\beta$ dla konstruktorów typów, dla których daje się określić rodzaj, zatrzymuje się.*

Dowód. Trywialne. Wynika z tego, że algorytm jest sterowany składnią, oraz z silnej normalizowalności kostruktorów typów. □

Twierdzenie 1. *Niech T_1 oraz T_2 będą konstruktorami typów, które daje się orodzajować. Wówczas jeśli $unify_\beta(T_1, T_2) = (\sigma, S)$, to (σ, S) jest β-unifikatorem konstruktorów typów T_1 i T_2.*

Dowód. Indukcja po głębokości rekursji.
Możliwe są następujące przypadki:

- $T_1 =_\beta T_2$ lub $T_1 \equiv \bar{X}$ lub $T_2 \equiv \bar{X}$
  dla tych przypadków teza zachodzi trywialnie.

- $T_1 \equiv \forall X_1 :: K_1.T_1'$ oraz $T_2 \equiv \forall X_2 :: K_2.T_2'$
  Wtedy z założenia indukcyjnego (σ_T, T) jest β-unifikatorem konstruktorów typów $\sigma_K\{X_1 :=$

$X_3\}T_1'$ oraz $\sigma_K\{X_2 := X_3\}T_2'$, gdzie σ_K jest unifikatorem rodzajów K_1 i K_2. Istnieje więc takie podstawienie schematowe θ oraz przemianowania α_1 i α_2, że zachodzą równości

$$\theta\alpha_1\sigma_K\{X_1 := X_3\}T_1' =_\beta T =_\beta \theta\alpha_2\sigma_K\{X_2 := X_3\}T_2'$$

$$\theta \setminus BTV(T) = \sigma_T$$

A wtedy

$$(\theta \circ \sigma_K)\alpha_1\{X_1 := X_3\}\forall X_1 :: K_1.T_1' = \forall X_3 :: \theta\sigma_K K_1.\theta\sigma_K\alpha_1\{X_1 := X_3\}T_1' =$$

$$= \forall X_3 :: (\theta \setminus BTV(T))\sigma_K K_1.\theta\alpha_1\sigma_K\{X_1 := X_3\}T_1' =_\beta \forall X_3 :: \sigma_T\sigma_K K_1.T$$

oraz

$$(\theta \circ \sigma_K) \setminus BTV(\forall X_3 :: \sigma_T\sigma_K K_1.T) = (\theta \setminus (\{X_3\} \cup BTV(T))) \circ \sigma_K =$$

$$= (\sigma_T \setminus X_3) \circ \sigma_K$$

Analogicznie pokażemy pozostałe dwie równości dla konstruktora T_2.
Zatem $((\sigma_T \setminus X_3) \circ \sigma_K, \forall X_3 :: \sigma_T\sigma_K K_1.T)$ jest β-unifikatorem konstruktorów T_1 i T_2.

- $T_1 \equiv \lambda X_1 :: K_1.T_1'$ oraz $T_2 \equiv \lambda X_2 :: K_2.T_2'$
  Dla tego przypadku dowód przeprowadzamy analogicznie do przypadku poprzedniego.

- $T_1 \equiv T_1' \to S_1$ oraz $T_2 \equiv T_2' \to S_2$
  Teza wynika wprost z założenia indukcyjnego oraz lematów 5 i 6.

- $T_1 \longrightarrow_\beta S_1$ lub $T_2 \longrightarrow_\beta S_2$
  Wynika z założenia indukcyjnego i z faktu, że β-redukcja zachowuje β-równość.

- $T_1 \equiv V_1\ S_1$ oraz $T_2 \equiv V_2\ S_2$
  Dowód przeprowadzamy analogicznie do przypadku z typem funkcji.

□

Fakt 2. *Algorytm $unify_\beta$ działający na konstruktorach typów nie zawierających zmiennych schematowych zwraca pusty unifikator jeśli konstruktory są β równe, w przeciwnym przypadku zawodzi (zawraca $fail$).*

Fakt 3. *Jeżeli w algorytmie pierwszy przypadek sprawdzający β-równość konstruktorów zamienimy na analogiczy sprawdzający tożsamość zmiennych typowych (a przy rozszerzeniach jeszcze typów bazowych), otrzymamy algorytm równoważny.*

4.4. Rekonstrukcja typów

4.4.1. Algorytm W

Dla rodzajów:
$kindof(\Gamma \vdash X ::?) =$
  let $\Omega\widehat{X}_1 \ldots \widehat{X}_n.K = \Gamma(X)$ in
  let $\widehat{Y}_1, \ldots, \widehat{Y}_n = fresh$ in
    $([], [\widehat{X}_1 := \widehat{Y}_1, \ldots, \widehat{X}_n := \widehat{Y}_n]K)$

$kindof(\Gamma \vdash T_1 \to T_2 ::?) =$
$\quad$ let $(\sigma_1, K_1) = kindof(\Gamma \vdash T_1 ::?)$ in
$\quad$ let $(\sigma_2, K_2) = kindof(\sigma_1\Gamma \vdash \sigma_1 T_1 ::?)$ in
$\quad$ let $\rho_1 = unify(\sigma_2 K_1, *)$ in
$\quad$ let $\rho_2 = unify(\rho_1 K_2, *)$ in
$\quad\quad (\rho_2 \circ \rho_1 \circ \sigma_2 \circ \sigma_1, *)$
$kindof(\Gamma \vdash \forall X :: K.T ::?) =$
$\quad$ let $(\sigma, K_T) = kindof(\Gamma, X :: K \vdash T ::?)$ in
$\quad$ let $\rho = unify(K_T, *)$ in
$\quad\quad (\rho \circ \sigma, *)$
$kindof(\Gamma \vdash \lambda X :: K.T ::?) =$
$\quad$ let $(\sigma, K_T) = kindof(\Gamma, X :: K \vdash T ::?)$ in
$\quad\quad (\sigma, \sigma K \Rightarrow K_T)$
$kindof(\Gamma \vdash T_1\ T_2 ::?) =$
$\quad$ let $(\sigma_1, K_1) = kindof(\Gamma \vdash T_1 ::?)$ in
$\quad$ let $(\sigma_2, K_2) = kindof(\sigma_1\Gamma \vdash \sigma_1 T_2 ::?)$ in
$\quad$ let $\widehat{X} = fresh$ in
$\quad$ let $\sigma_3 = unify(\sigma_2 K_1, K_2 \Rightarrow \widehat{X})$ in
$\quad\quad (\sigma_3 \circ \sigma_2 \circ \sigma_1, \sigma_3 \widehat{X})$
$kindof(\Gamma \vdash \bar{X} ::?) =$
$\quad\quad ([], *)$

Dla typów:
$typeof(\Gamma \vdash x :?) =$
$\quad$ let $\Omega \bar{X}_1 \ldots \bar{X}_n \widehat{X}_1 \ldots \widehat{X}_m.T = \Gamma(x)$ in
$\quad$ let $\bar{Y}_1, \ldots, \bar{Y}_n, \widehat{Y}_1, \ldots, \widehat{Y}_m = fresh$ in
$\quad\quad ([], [\bar{X}_1 := \bar{Y}_1, \ldots, \bar{X}_n := \bar{Y}_n, \widehat{X}_1 := \widehat{Y}_1, \ldots, \widehat{X}_m := \widehat{Y}_m]T)$
$typeof(\Gamma \vdash \lambda x : T.t :?) =$
$\quad$ let $(\sigma_1, K) = kindof(\Gamma \vdash T ::?)$ in
$\quad$ let $\sigma_2 = unify(K, *)$ in
$\quad$ let $(\sigma_3, T_t) = typeof(\sigma_2\sigma_1(\Gamma, x : T) \vdash \sigma_2\sigma_1 t :?)$ in
$\quad\quad (\sigma_3 \circ \sigma_2 \circ \sigma_1, \sigma_3\sigma_2\sigma_1 T \to T_t)$
$typeof(\Gamma \vdash t_1\ t_2 :?) =$
$\quad$ let $(\sigma_1, T_1) = typeof(\Gamma \vdash t_1 :?)$ in
$\quad$ let $(\sigma_2, T_2) = typeof(\sigma_1\Gamma \vdash \sigma_1 t_2 :?)$ in
$\quad$ let $\bar{X} = fresh$ in
$\quad$ let $(\sigma_3, T_2' \to T_3) = unify_\beta(\sigma_2 T_1, T_2 \to \bar{X})$ in
$\quad\quad (\sigma_3 \circ \sigma_2 \circ \sigma_1, T_3)$
$typeof(\Gamma \vdash \lambda X :: K.t :?) =$
$\quad$ let $(\sigma, T) = typeof(\Gamma, X :: K \vdash t :?)$ in
$\quad\quad (\sigma \setminus X, \forall X :: \sigma K.T)$

$typeof(\Gamma \vdash t[T] :?) =$
  let $(\sigma_1, T_t) = typeof(\Gamma \vdash t :?)$ in
  let $(\sigma_2, K) = kindof(\sigma_1\Gamma \vdash \sigma_1 T ::?)$ in
  let $\bar{X} = fresh$ in
  let $Y = fresh$ in
  let $(\sigma_3, \forall Z :: K'.T') = unify_\beta(\sigma_2 T_t, \forall Y :: K.\bar{X})$in
    $(\sigma_3 \circ \sigma_2 \circ \sigma_1, \{Y := \sigma_3\sigma_2\sigma_1 T\}T')$
$typeof(\Gamma \vdash \text{let } x = t_1 \text{ in } t_2 :?) =$
  let $(\sigma_1, T_1) = typeof(\Gamma \vdash t_1 :?)$ in
  let $\left\{\bar{X}_1, \ldots, \bar{X}_n\right\} = \bar{Var}(T_1) \setminus \bar{Var}(\Gamma)$ in
  let $\left\{\widehat{X}_1, \ldots, \widehat{X}_m\right\} = \widehat{Var}(T_1) \setminus \widehat{Var}(\Gamma)$ in
  let $(\sigma_2, T_2) = typeof(\sigma_1\Gamma, x : \Omega\bar{X}_1 \ldots \bar{X}_n\widehat{X}_1 \ldots \widehat{X}_m.T_2 \vdash \sigma_1 t_2 :?)$ in
    $(\sigma_2 \circ \sigma_1, T_2)$
$typeof(\Gamma \vdash \text{tlet } X = T \text{ in } t :?) =$
  let $(\sigma_1, K) = kindof(\Gamma \vdash T ::?)$ in
  let $(\sigma_2, T_t) = typeof(\sigma_1\Gamma \vdash \{X(\widehat{Var}(K) \setminus \widehat{Var}(\Gamma)) := \sigma_1 T\}\sigma_1 t :?)$ in
    $(\sigma_2 \circ \sigma_1, T_t)$

4.4.2. Własności

Twierdzenie 2. *Funkcja typeof zatrzymuje się.*

Twierdzenie 3. *Niech t i Γ nie zawierają zmiennych schematowych. Wówczas $typeof(\Gamma \vdash t :?) = (\sigma, T)$ wtedy i tylko wtedy, gdy $\Gamma \vdash t : T'$, gdzie $T' =_\beta T$ oraz $\sigma = []$ (tzn. gdy program zawiera wszystkie anotacje, algorytm typeof wykonuje zwykłą kontrolę typów w systemie F_ω).*

Dowód. Szkic dowodu:

1. Indukcyjnie po strukturze typu pokazujemy analogiczne twierdzenie dla rodzajów i funkcji $kindof$:
   Niech T i Γ nie zawierają zmiennych schematowych. Wówczas $kindof(\Gamma \vdash t ::?) = (\sigma, K)$ wtedy i tylko wtedy gdy $\Gamma \vdash T :: K$ oraz $\sigma = []$.

2. Indukcyjnie po strukturze wyrażenia t pokażemy implikację tezy w jedną stronę.

3. Pokazujemy indukcyjnie, że drzewa wyprowadzenia $\Gamma \vdash t : T$ można zawsze przebudować tak, by z reguły
   $$\frac{\Gamma \vdash t : S \quad S \equiv T \quad \Gamma \vdash T :: *}{\Gamma \vdash t : T}$$
   korzystać jedynie nad argumentem aplikacji i w korzeniu drzewa wyprowadzenia.

4. Indukcyjnie po tak przebudowanym drzewie pokażemy implikację tezy w drugą stronę.

□

Twierdzenie 4. *Niech t i Γ będą takim termem i kontekstem, że $typeof(\Gamma \vdash t :?) = (\sigma, T)$. Wówczas istnieje takie podstawienie schematowe θ, że zachodzą następujące własności:*

- $\theta\Gamma \vdash \theta t : T$

- $\theta \setminus BTV(t) = \sigma$

Dowód. Szkic dowodu:

1. Podobnie jak w dowodzie twierdzenia 3 najpierw indukcyjnie dowodzimy analogicznego twierdzenia dla funkcji *kindof* i relacji rodzajowania.

2. Indukcyjnie po strukturze termu t pokazujemy tezę. Podstawienie schematowe θ konstruujemy tak jak podstawienie schematowe σ tyle, że z pominięciem anulowania zmiennych związanych. W miejscach w których korzystamy z β-unifikacji korzystamy z definicji β-unifikatora by skonstruować podstawienie schematowe θ.

□

5. Inne własności F_ω

5.1. Nierozstrzygalność

Fakt 4. *Nierozstrzygalne są problemy:*

- *sprawdzania typu: dane* Γ, M, τ*, pytamy czy* $\Gamma \vdash M : \tau$
- *typowalność: dane M, pytamy czy* $\exists \Gamma \tau . \Gamma \vdash M : \tau$
- *sprawdzania niepustości typu*

5.2. Redukcja $\Rightarrow$ na typach - definicja i własności

Jak już wspomnieliśmy, możemy na typach zdefiniować redukcję, która przebiega bardzo podobnie do β-redukcji na termach.

$$\frac{}{\mathtt{T} \Rightarrow \mathtt{T}} \qquad \frac{\mathtt{S_1} \Rightarrow \mathtt{T_1} \quad \mathtt{S_2} \Rightarrow \mathtt{T_2}}{\mathtt{S_1} \to \mathtt{S_2} \Rightarrow \mathtt{T_1} \to \mathtt{T_2}}$$

$$\frac{\mathtt{S_1} \Rightarrow \mathtt{T_1} \quad \mathtt{S_2} \Rightarrow \mathtt{T_2}}{\mathtt{S_1\ S_2} \Rightarrow \mathtt{T_1\ T_2}} \qquad \frac{\mathtt{S} \Rightarrow \mathtt{T}}{\forall \mathtt{X} :: \mathtt{K.S} \Rightarrow \forall \mathtt{X} :: \mathtt{K.T}}$$

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

$$\frac{\mathtt{S_1} \Rightarrow \mathtt{T_1} \quad \mathtt{S_2} \Rightarrow \mathtt{T_2}}{(\lambda \mathtt{X} :: \mathtt{K.S_1})\mathtt{S_2} \Rightarrow [\mathtt{X} \mapsto \mathtt{T_2}]\mathtt{T_1}}$$

Lemat 7. *Jeśli* $S \Rightarrow T$ *i* $S \Rightarrow U$*, to istnieje* P *takie, że* $T \Rightarrow P$ *i* $U \Rightarrow P$*.*

Lemat 8. *Zachodzi własność Churcha-Rossera dla* $\Rightarrow$*, czyli jeśli* $S \Rightarrow^* T$ *i* $S \Rightarrow^* U$*, to istnieje* P *takie, że* $T \Rightarrow^* P$ *i* $U \Rightarrow^* P$*.*

Lemat 9. $S \equiv T$ *zachodzi wtedy i tylko wtedy, gdy* $S \Leftrightarrow^* T$*.*

5.3. Inne własności

Twierdzenie 5. *Jeżeli* $\Gamma \vdash t : T$ *i* $t \to t'$ *to* $t' : T$.

Lemat 10. *Jeśli* t *jest zamkniętą wartością i* $\vdash t : T_1 \to T_2$, *to* t *jest abstrakcją.*

Lemat 11. *Jeśli* t *jest zamkniętą wartością i* $\vdash t : \forall X : T_1.T_2$, *to* t *jest abstrakcją typową.*

Twierdzenie 6. *Jeżeli* $\vdash t : T$ *dla pewnego* T, *to albo* t *jest wartością, albo jakiś* t' *taki, że* $t \to t'$.

Literatura

[1] Pierce B., *Types and programing languages*, The MIT Press, 2002

[2] Urzyczyn P., *Materiały do wykładu Rachunek Lambda*

[3] Mazurak K.,Zhao J.,Zdancevic S., *Lightweight Linear Types in System F°*