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 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*
	$\lambda\mathtt{x}:\mathtt{T.t}$	*abstrakcja anotowana*
	$\lambda\mathtt{x.t}$	*abstrakcja nieanotowana*
	$\mathtt{t\,t}$	*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 $\mathtt{Tb} = \lambda\mathtt{X.X} \to \mathtt{Bool}$ i $\lambda\mathtt{X.X}$ są abstrakcjami konstruktowymi, ale $\lambda\mathtt{x.x}$ jest abstrakcją na termach. Do konstruktora $\mathtt{Tb}$ możemy zaaplikować $\mathtt{Bool}$ i dostaniemy $(\lambda\mathtt{X.X} \to \mathtt{Bool})\mathtt{Bool}$ 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 $(\mathtt{Tb^n})\mathtt{Bool}$ dla n naturalnego, a $\mathtt{T^n}$ 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 2.2.3

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

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

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

- rodzaje

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

- abstrakcję i aplikację typową na typach

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

Czy moglibyśmy napisać $\lambda\mathtt{X} :: * \Rightarrow *.\lambda\mathtt{x} : \mathtt{X}.\mathtt{x} \;:\; \forall\mathtt{X} :: * \Rightarrow *.\mathtt{X} \rightarrow \mathtt{X} \;:: * \Rightarrow *$? 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

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

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

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.

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

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

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

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 $*$:

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

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 $\mathtt{t} : (\lambda \mathtt{X}.\mathtt{X})\mathtt{Nat}$ otrzymamy:

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

Nie musimy również pisać reguł typu:

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

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

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

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

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

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:

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

a sekwencje definiowaliśmy jako:

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

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

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

W rachunku F_ω pojawia się możliwość zakodowania unit i Unit. Robimy to w taki sposób:

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

$\mathtt{Unit} \stackrel{\text{def}}{=} \forall \mathtt{X} :: *.\mathtt{X} \to \mathtt{X}$

3.3. Anotacje typowe

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

$$\begin{array}{lll} \mathtt{t} ::= & \cdots & \textit{termy} \\ & \mathtt{t\ as\ T} & \textit{anotacja typowa} \end{array}$$

Ewaluacja i typowanie nie zmieniają się.

3.4. Definicje lokalne

$$\mathtt{let}\ \mathtt{x} = \mathtt{t_1}\ \mathtt{in}\ \mathtt{t_2} \overset{\mathtt{def}}{=} (\lambda \mathtt{x} : \mathtt{T}.\mathtt{t_2})\mathtt{t_1} \qquad \text{gdzie } \mathtt{t_1} : \mathtt{T}$$

3.5. Rekordy

Składnię rekordów zdefiniujemy następująco:

$$\begin{array}{lcr} \mathtt{t} ::= & \ldots & \textit{termy} \\ & \{\mathtt{l_i} = \mathtt{t_i}\ ^{\mathtt{i}\in 1..\mathtt{n}}\} & \textit{rekord} \\ & \mathtt{t.l} & \textit{projekcja} \\ \\ \mathtt{T} ::= & \ldots & \textit{typy} \\ & \{\mathtt{l_i} : \mathtt{T_i}\ ^{\mathtt{i}\in 1..\mathtt{n}}\} & \textit{typ rekordu} \\ \\ \mathtt{v} ::= & \ldots & \textit{wartości} \\ & \{\mathtt{l_i} = \mathtt{v_i}\ ^{\mathtt{i}\in 1..\mathtt{n}}\} & \textit{wartość rekordu} \end{array}$$

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

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

oraz wprowadzimy niewielkie zmiany w regułach typowania:

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

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

a ewaluację pozostawimy bez zmian:

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

$$\frac{\mathtt{t_i} \longrightarrow \mathtt{t_i'}}{\{\mathtt{l_1} = \mathtt{v_1}, \ldots, \mathtt{l_{i-1}} = \mathtt{v_{i-1}}, \mathtt{l_i} = \mathtt{t_i}, \ldots, \mathtt{l_n} = \mathtt{t_n}\} \longrightarrow \{\mathtt{l_1} = \mathtt{v_1}, \ldots, \mathtt{l_{i-1}} = \mathtt{v_{i-1}}, \mathtt{l_i} = \mathtt{t_i'}, \ldots, \mathtt{l_n} = \mathtt{t_n}\}}$$

3.6. Warianty

Składnię wariantów zdefiniujemy następująco:

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

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

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

a w regułąch typowania wprowadzimy małe zmiany:

$$\frac{\Gamma \vdash \mathtt{t_0} : < \mathtt{l_i} : \mathtt{T_i}^{\ \mathtt{i} \in 1..\mathtt{n}} > \qquad \Gamma, \mathtt{x_1} : \mathtt{T_1} \vdash \mathtt{t_1} : \mathtt{T} \ \ldots\ \Gamma, \mathtt{x_n} : \mathtt{T_n} \vdash \mathtt{t_n} : \mathtt{T}}{\Gamma \vdash \mathtt{case\ t_0\ of}\ \ {< \mathtt{l_i} = \mathtt{x_i} > \Rightarrow \mathtt{t_i}}^{\ \mathtt{i} \in 1..\mathtt{n}} \ : \mathtt{T}}$$

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

natomiast ewaluacja pozostanie bez zmian:

$$\mathtt{case}\ (< \mathtt{l_j} = \mathtt{t_j} >\ \mathtt{as\ T})\ \mathtt{of}\ \ {< \mathtt{l_i} = \mathtt{x_i} > \Rightarrow \mathtt{t_i}}^{\ \mathtt{i} \in 1..\mathtt{n}} \longrightarrow [\mathtt{x_j} \mapsto \mathtt{v_j}]\mathtt{t_j}$$

$$\frac{\mathtt{t} \longrightarrow \mathtt{t}'}{\mathtt{case\ t\ of}\ \ {< \mathtt{l_i} = \mathtt{x_i} > \Rightarrow \mathtt{t_i}}^{\ \mathtt{i} \in 1..\mathtt{n}} \longrightarrow \mathtt{case\ t'\ of}\ \ {< \mathtt{l_i} = \mathtt{x_i} > \Rightarrow \mathtt{t_i}}^{\ \mathtt{i} \in 1..\mathtt{n}}}$$

$$\frac{\mathtt{t_j} \longrightarrow \mathtt{t}'_\mathtt{j}}{< \mathtt{l_j} = \mathtt{t_j} >\ \mathtt{as\ T} \longrightarrow < \mathtt{l_j} = \mathtt{t}'_\mathtt{j} >\ \mathtt{as\ T}}$$

3.7. Punkt stały

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

Typowanie

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

Ewaluacja

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

3.8. 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*
	$\mathtt{List\ T}$	*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:

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

Typowanie:

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

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

Ewaluacja:

$$\mathtt{aaa}$$

3.9. 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*
	$\{{}^*\mathtt{T} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as\ T}$	*pakowanie*
	$\mathtt{let}\ \{\mathtt{X}, \mathtt{x}\} = \mathtt{t\ in\ t}$	*odpakowanie*
$\mathtt{T} ::=$	$\cdots$	*typy*
	$\{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$	*typ egzystencjalny*
$\mathtt{v} ::=$	$\cdots$	*wartości*
	$\{{}^*\mathtt{T}, \mathtt{v}\}\ \mathtt{as\ T}$	*pakowanie*

W systemie F_ω powyższe elementy języka możemy zdefiniować następująco:

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

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

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

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

$\{{}^*\mathtt{U} :: \mathtt{K}, \mathtt{t}\}\ \mathtt{as}\ \{\exists \mathtt{X} :: \mathtt{K}, \mathtt{T}\}$

$= \mathtt{let\ x = t\ in}\ \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}]\mathtt{x} =$

$= (\lambda \mathtt{x} : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{T}.\lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}]\mathtt{x})\mathtt{t} =$
$\overset{\mathtt{t}:[\mathtt{X}\mapsto\mathtt{U}]\mathtt{T}}{=} \lambda \mathtt{Y} :: *.\lambda \mathtt{f} : (\forall \mathtt{X} :: \mathtt{K}.\mathtt{T} \to \mathtt{Y}).\mathtt{f}[\mathtt{U}](\mathtt{t} : [\mathtt{X} \mapsto \mathtt{U}]\mathtt{T})$
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.

Tworzenie rodzaju:

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

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:

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

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

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

3.10. Typy rekurencyjne

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

`t` ::=	...	*termy*
	`fold`[`T`] `t`	*folding*
	`unfold`[`T`] `t`	*unfolding*
`T` ::=	...	*typy*
	μ`X.T`	*typ rekursywny*
`v` ::=	...	*wartości*
	`fold`[`T`] `v`	*folding*

Tworzenie rodzaju:

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

Typowanie:

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

Ewaluacja:

3.11. dopasowanie wzorca

4. Rekonstrukcja typów

4.1. Kilka słów o składni.

Rozważmy język o składni z 2 rozdziału 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)*

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:

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

I analogicznie dla rodzajów.

Teraz sam kontekst typowania ma postać:

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

4.2. 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, 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 częściowe, tzn. samo podstawienie oprócz przyporządtkowań postaci $[\bar{X} := T]$, może również zawierać podstawienia postaci $[\bar{X} := fail]$. Oraz dodając jeszcze operację anulowania podstawienia:

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

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

Definicja 1. *β-unifikatorem* dla konstruktorów typów T_1 i T_2 nazwiemy taką parę (σ, S), że istnieją podstawienia θ_1 i θ_2 takie, że:

$$\theta_1 T_1 =_\beta S =_\beta \theta_2 T_2$$

oraz

$$\theta_1 \setminus BTV(T_1) = \sigma = \theta_2 \setminus BTV(T_2)$$

Definicja 2. Powiemy, że β-unifikator (σ, S) jest *ogólniejszy* od β-unifikatora (θ, T), jeżeli istnieje takie podstawienie ρ i przemianowanie α, że

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

Definicja 3. *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 1. *Niech A i B będą podzbiorami zbioru zmiennych typowych kwnatyfikowanych abstrakcją, niech θ będzie podstawieniem. Wówczas zachodzi*

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

Dowód. □

Lemat 2. *Niech A będzie podzbiorem zbioru zmiennych typowych kwnatyfikowanych abstrakcją, niech θ i ρ będą podstawieniami, oraz niech zachodzi $Var(\rho) \cap A = \emptyset$. Wówczas zachodzi*

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

.

Dowód. □

Lemat 3. *Niech A będzie podzbiorem zbioru zmiennych typowych kwnatyfikowanych abstrakcją, niech θ będzie dowolnym podstawieniem, niech ρ będzie podstawieniem takim, że $Var(\rho) \cap A = \emptyset$ (w szczególności zawierającym tylko zmienne rodzajowe). Wówczas zachodzi*

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

Dowód. □

Lemat 4. *Niech (σ, S) będzie β-unifikatorem dla konstruktorów typów T_1 i T_2, oraz niech ρ będzie podstawieniem takim, że $Var(\rho) \cap BTV(T_1) = \emptyset$ oraz $Var(\rho) \cap BTV(T_2) = \emptyset$. Wówczas $(\rho \circ \sigma, \rho S)$ również jest β-unifikatorem dla konstruktorów typów T_1 i T_2.*

Dowód. Wiemy, że istnieją takie θ_1 i θ_2, że

$$\theta_1 T_1 =_\beta S =_\beta \theta_2 T_2 \quad \text{oraz} \quad \theta_1 \setminus BTV(T_1) = \sigma = \theta_2 \setminus BTV(T_2)$$

A wtedy zachodzi

$$(\rho \circ \theta_1) T_1 = \rho(\theta_1 T_1) =_\beta \rho S =_\beta \rho(\theta_2 T_2) = (\rho \circ \theta_2) T_2$$

oraz z lematu 2

$$(\rho \circ \theta_1) \setminus BTV(T_1) = \rho \circ (\theta_1 \setminus BTV(T_1)) = \rho \circ \sigma$$
$$(\rho \circ \theta_2) \setminus BTV(T_2) = \rho \circ (\theta_2 \setminus BTV(T_2)) = \rho \circ \sigma$$

□

Lemat 5. *Niech* T_1 *i* T_2 *będą konstrktorami typów,* ρ *niech będzie podstawieniem takim, że* $Var(\rho) \cap BTV(T_1) = \emptyset$ *oraz* $Var(\rho) \cap BTV(T_2) = \emptyset$*, 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_i, że

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

A wtedy

$$(\theta_i \circ \rho) T_i =_\beta S$$

oraz z lematu 3 i z tego że $BTV(\rho) \cap BTV(\theta_i) = \emptyset$ mamy

$$(\theta_i \circ \rho) \setminus BTV(T_i) = (\theta_i \circ BTV(T_i)) \circ \rho = (\theta_i \circ BTV(\rho T_i)) \circ \rho = \sigma \circ \rho$$

Co kończy dowód. $\square$

4.2.1. 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 $(\sigma_T, T) = unify_\beta(\sigma_K T_1, \sigma_K\{X_2 := X_1\}T_2)$ in
    $((\sigma_T \setminus X_1) \circ \sigma_K, \forall X_1 :: \sigma_T \sigma_K K_1.T)$
$unify_\beta(\lambda X_1 :: K_1.T_1, \lambda X_2 :: K_2.T_2)$ =
  let $\sigma_K = unify(K_1, K_2)$ in
  let $(\sigma_T, T) = unify_\beta(\sigma_K T_1, \sigma_K\{X_2 := X_1\}T_2)$ in
    $((\sigma_T \setminus X_1) \circ \sigma_K, \lambda X_1 :: \sigma_T \sigma_K K_1.T)$
$unify_\beta(T_1 \to S_1, T_2 \to S_2)$ =
  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
    $(\sigma \circ \theta, \sigma V\ T)$

4.2.2. 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 głebokości 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 załorzenia indukcyjnego (σ_T, T) jest β-unifikatorem konstruktorów typów $\sigma_K T_1'$ oraz $\sigma_K\{X_2 := X_1\}T_2'$, gdzie σ_K jest unifikatorem rodzajów K_1 i K_2. Istnieją więc takie podstawienia θ_1 i θ_2, że zachodzą równości

  $$\theta_1\sigma_K T_1' =_\beta T =_\beta \theta_2\sigma_K\{X_2 := X_1\}T_2'$$

  $$\theta_1 \setminus BTV(\sigma_K T_1') = \sigma_T = \theta_2 \setminus BTV(\sigma_K\{X_2 := X_1\}T_2')$$

  A wtedy

  $$(\theta_1 \circ \sigma_K)\forall X_1 :: K_1.T_1' = \forall X_1 :: \theta_1\sigma_K K_1.\theta_1\sigma_K T_1' =_\beta$$

  $$=_\beta \forall X_1 :: (\theta_1 \setminus BTV(\sigma_K T_1'))\sigma_K K_1.T = \forall X_1 :: \sigma_T\sigma_K K_1.T$$

  oraz

  $$(\theta_1 \circ \sigma_K) \setminus BTV(\forall X_1 :: K_1.T_1') = (\theta_1 \setminus (\{X_1\} \cup BTV(\sigma_K T_1'))) \circ \sigma_K =$$

  $$= (\sigma_T \setminus X_1) \circ \sigma_K$$

  Analogicznie pokarzemy pozostałe dwie równości dla konstruktora T_2. Zatem $((\sigma_T \setminus X_1) \circ \sigma_K, \forall X_1 :: \sigma_T\sigma_K K_1.T)$ 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 załorzenia indukcyjnego oraz lematów 4 i 5.

- $T_1 \longrightarrow_\beta S_1$ lub $T_2 \longrightarrow_\beta S_2$
  Wynika z załorzenia 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* $fail$*).*

Dowód. □

Fakt 3. *Jeżeli z algorytmu usuniemy pierwszy przypadek sprawdzający* β*-równość konstruktorów, otrzymamy algorytm równoważny.*

Dowód. □

Tu powinno pojawić się jeszcze kilka dowodów własności.

4.3. Rekonstrukcja typów

4.3.1. 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
    $([], [\widehat{X}_1 := \widehat{Y}_1, \ldots, \widehat{X}_n := \widehat{Y}_n]K)$
$kindof(\Gamma \vdash T_1 \to T_2 ::?) =$
  let $(\sigma_1, K_1) = kindof(\Gamma \vdash T_1 ::?)$ in
  let $(\sigma_2, K_2) = kindof(\sigma_1\Gamma \vdash \sigma_1 T_1 ::?)$ in
  let $\rho_1 = unify(\sigma_2 K_1, *)$ in
  let $\rho_2 = unify(\rho_1 K_2, *)$ in
    $(\rho_2 \circ \rho_1 \circ \sigma_2 \circ \sigma_1, *)$
$kindof(\Gamma \vdash \forall X :: K.T ::?) =$
  let $(\sigma, K_T) = kindof(\Gamma, X :: K \vdash T ::?)$ in
  let $\rho = unify(K_T, *)$ in
    $(\rho \circ \sigma, *)$
$kindof(\Gamma \vdash \lambda X :: K.T ::?) =$
  let $(\sigma, K_T) = kindof(\Gamma, X :: K \vdash T ::?)$ in
    $(\sigma, K \Rightarrow K_T)$
$kindof(\Gamma \vdash T_1\ T_2 ::?) =$
  let $(\sigma_1, K_1) = kindof(\Gamma \vdash T_1 ::?)$ in
  let $(\sigma_2, K_2) = kindof(\sigma_1\Gamma \vdash \sigma_1 T_2 ::?)$ in
  let $\widehat{X} = fresh$ in
  let $\sigma_3 = unify(\sigma_2 K_1, K_2 \Rightarrow \widehat{X})$ in
    $(\sigma_3 \circ \sigma_2 \circ \sigma_1, \sigma_3\widehat{X})$
$kindof(\Gamma \vdash \bar{X} ::?) =$
    $([], *)$

Dla typów:

$typeof(\Gamma \vdash x :?) =$
  let $\Omega\bar{X}_1 \ldots \bar{X}_n\widehat{X}_1 \ldots \widehat{X}_m.T = \Gamma(x)$ in
  let $\bar{Y}_1, \ldots, \bar{Y}_n, \widehat{Y}_1, \ldots, \widehat{Y}_m = fresh$ in
    $([], [\bar{X}_1 := \bar{Y}_1, \ldots, \bar{X}_n := \bar{Y}_n, \widehat{X}_1 := \widehat{Y}_1, \ldots, \widehat{X}_m := \widehat{Y}_m]T)$

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

Tu można wspomnieć o jego własnościach.

5. Własności i dowody

5.1. Inne własności F_ω

Definicja 4. 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 $\exists \Gamma \tau . \Gamma \vdash M : \tau$

5.2. pare słów o rozszerzeniach

6. Praktyczne zastosowanie

7. Podsumowanie

Literatura

[1] Pierce,