Вывод типов от Хиндли — Милнера до GHC 8.8

Лекции 3–5

В. Н. Брагилевский

3 марта 2019 г.

Computer Science клуб (Санкт-Петербург)

Институт математики, механики и компьютерных наук
имени И. И. Воровича, Южный федеральный университет (Ростов-на-Дону)

Я прочитал эту статью специально для вас

D. Vytiniotis, S. Peyton Jones, T. Schrijvers, M. Sulzmann.

OutsideIn(x) modular type inference with local assumptions.

Обожаю аннотации статей от SPJ

Advanced type system features, such as GADTs, type classes, and type families have proven to be invaluable language extensions for ensuring data invariants and program correctness among others. Unfortunately, they pose a tough problem for type inference, because they introduce local type assumptions.

In this article we present a novel constraint-based type inference approach for local type assumptions. Our system, called OutsideIn(X), is parameterised over the particular underlying constraint domain X, in the same way as HM(X). This stratification allows us to use a common metatheory and inference algorithm.

Going beyond the general framework, we also give a particular constraint solver for X = type classes + GADTs + type families, a non-trivial challenge in its own right.

Обожаю аннотации статей от SPJ

Advanced type system features, such as GADTs, type classes, and type families have proven to be invaluable language extensions for ensuring data invariants and program correctness among others. Unfortunately, they pose a tough problem for type inference, because they introduce local type assumptions.

In this article we present a novel constraint-based type inference approach for local type assumptions. Our system, called OutsideIn(X), is parameterised over the particular underlying constraint domain X, in the same way as HM(X). This stratification allows us to use a common metatheory and inference algorithm.

Going beyond the general framework, we also give a particular constraint solver for X = type classes + GADTs + type families, a non-trivial challenge in its own right.

Обожаю аннотации статей от SPJ

Advanced type system features, such as GADTs, type classes, and type families have proven to be invaluable language extensions for ensuring data invariants and program correctness among others. Unfortunately, they pose a tough problem for type inference, because they introduce local type assumptions.

In this article we present a novel constraint-based type inference approach for local type assumptions. Our system, called OutsideIn(X), is parameterised over the particular underlying constraint domain X, in the same way as HM(X). This stratification allows us to use a common metatheory and inference algorithm.

Going beyond the general framework, we also give a particular constraint solver for X = type classes + GADTs + type families, a non-trivial challenge in its own right.

Обожаю аннотации статей от SPJ

Advanced type system features, such as GADTs, type classes, and type families have proven to be invaluable language extensions for ensuring data invariants and program correctness among others. Unfortunately, they pose a tough problem for type inference, because they introduce local type assumptions.

In this article we present a novel constraint-based type inference approach for local type assumptions. Our system, called OutsideIn(X), is parameterised over the particular underlying constraint domain X, in the same way as HM(X). This stratification allows us to use a common metatheory and inference algorithm.

Going beyond the general framework, we also give a particular constraint solver for X = type classes + GADTs + type families, a non-trivial challenge in its own right.

Обожаю аннотации статей от SPJ

Advanced type system features, such as GADTs, type classes, and type families have proven to be invaluable language extensions for ensuring data invariants and program correctness among others. Unfortunately, they pose a tough problem for type inference, because they introduce local type assumptions.

In this article we present a novel constraint-based type inference approach for local type assumptions. Our system, called OutsideIn(X), is parameterised over the particular underlying constraint domain X, in the same way as HM(X). This stratification allows us to use a common metatheory and inference algorithm.

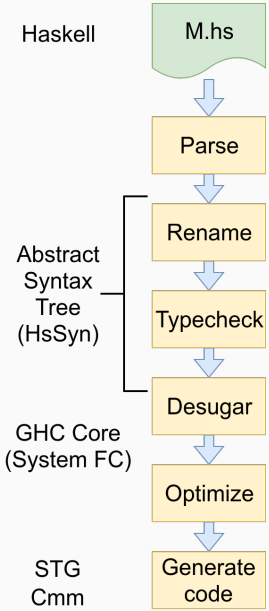
Going beyond the general framework, we also give a particular constraint solver for X = type classes + GADTs + type families, a non-trivial challenge in its own right.

Обожаю аннотации статей от SPJ

Advanced type system features, such as GADTs, type classes, and type families have proven to be invaluable language extensions for ensuring data invariants and program correctness among others. Unfortunately, they pose a tough problem for type inference, because they introduce local type assumptions.

In this article we present a novel constraint-based type inference approach for local type assumptions. Our system, called OutsideIn(X), is parameterised over the particular underlying constraint domain X, in the same way as HM(X). This stratification allows us to use a common metatheory and inference algorithm.

Going beyond the general framework, we also give a particular constraint solver for X = type classes + GADTs + type families, a non-trivial challenge in its own right.



⇐ Мы находимся здесь

Трудности вывода типов в Haskell

Единственное типовое ограничение в ХМ — равенство типов

Применение функции типа $\tau_1 \to \tau_2$ к аргументу типа τ_3 порождает ограничение $\tau_1 \sim \tau_3$.

В Haskell есть много чего ещё, например, классы типов

```
member :: Eq a => a -> [a] -> Bool
```

Ограничение (по месту вызова `member`): $\mathrm{Eq}\ \tau$

В Haskell есть много чего ещё, например, классы типов

```
member :: Eq a => a -> [a] -> Bool
```

Ограничение (по месту вызова `member`): $\mathrm{Eq}\ \tau$

Вот аксиома, которая может удовлетворить ограничению:

```
instance Eq Int
```

Как насчёт MPTC и FD?

Из $C\ \tau\ v_1 \wedge C\ \tau\ v_2$ вытекает $v_1 \sim v_2$

Работоспособность ХМ зависит от существования главных типов

```
let f x = <rhs>
in <body>
```

Например, `reverse` $:: \forall a.[a] \to [a]$

Работоспособность XM зависит от существования главных типов

```
let f x = <rhs>
in <body>
```

Например, `reverse` $:: \forall a.[a] \to [a]$

$$\frac{e \vdash t : S \qquad e, x : S \vdash u : [B]}{e \vdash \textbf{let}\ x = t\ \textbf{in}\ u : [B]}$$

К сожалению, в присутствии GADT главные типы могут пропадать

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a

test (T1 n) _ = n > 0
test T2 r = r
```

Каков тип `test`?

К сожалению, в присутствии GADT главные типы могут пропадать

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a

test (T1 n) _ = n > 0
test T2 r = r
```

Каков тип `test`?

$\texttt{test} :: \forall a.\mathrm{T}\ a \to \mathrm{Bool} \to \mathrm{Bool}$

К сожалению, в присутствии GADT главные типы могут пропадать

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a

test (T1 n) _ = n > 0  -- T a ~ T Bool
test T2 r = r
```

Каков тип `test`?

`test` : : $\forall a.\mathrm{T}\ a \to \mathrm{Bool} \to \mathrm{Bool}$

`test` : : $\forall a.\mathrm{T}\ a \to a \to a$

К сожалению, в присутствии GADT главные типы могут пропадать

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a

test (T1 n) _ = n > 0  -- T a ~ T Bool
test T2 r = r
```

Каков тип `test`?

`test` : : $\forall a.\mathrm{T}\ a \to \mathrm{Bool} \to \mathrm{Bool}$

`test` : : $\forall a.\mathrm{T}\ a \to a \to a$

Нужно реджектить!

Но действовать следует аккуратно, ведь GADT — не приговор

```
test2 (T1 n) _ = n > 0
test2 T2 r = not r

ghci> :type test2
test2 :: T a -> Bool -> Bool
```

Экземпляры классов типов создают сложные схемы аксиом

```
class Foo a b where
  foo :: a -> b -> Int

instance Foo Int b

instance Foo a b => Foo [a] b
```

Но это снова может приводить к пропаданию главных типов

```
g y = let h :: forall c. c -> Int
          h x = foo y x
      in h True
```

Но это снова может приводить к пропаданию главных типов

```
g y = let h :: forall c. c -> Int
          h x = foo y x
      in h True

g :: Int -> Int -- Foo Int b
```

Но это снова может приводить к пропаданию главных типов

```
g y = let h :: forall c. c -> Int
          h x = foo y x
      in h True
```

```
g :: Int -> Int -- Foo Int b
g :: [Int] -> Int -- Foo a b => Foo [a] b
```

Но это снова может приводить к пропаданию главных типов

```
g y = let h :: forall c. c -> Int
          h x = foo y x
      in h True
```

```
g :: Int -> Int -- Foo Int b
g :: [Int] -> Int -- Foo a b => Foo [a] b
g :: [[Int]] -> Int -- Foo a b => Foo [a] b
```

Но это снова может приводить к пропаданию главных типов

```
g y = let h :: forall c. c -> Int
          h x = foo y x
      in h True
```

```
g :: Int -> Int -- Foo Int b
g :: [Int] -> Int -- Foo a b => Foo [a] b
g :: [[Int]] -> Int -- Foo a b => Foo [a] b
g :: . . .
```

Усложнение системы типов позволяет вернуть главные типы

$$\texttt{test}:: \forall ab.(a \sim \text{Bool} \supset b \sim \text{Bool}) \Rightarrow T\ a \to b \to b$$

$$\texttt{g}:: \forall b.(\forall c.\text{Foo}\ b\ c) \Rightarrow b \to Int$$

Усложнение системы типов позволяет вернуть главные типы

$$\texttt{test}:: \forall ab.(a \sim \text{Bool} \supset b \sim \text{Bool}) \Rightarrow T\ a \to b \to b$$

$$\texttt{g}:: \forall b.(\forall c.\text{Foo}\ b\ c) \Rightarrow b \to Int$$

НО НЕ ОЧЕНЬ ХОЧЕТСЯ!!!

Основанная на ограничениях система типов

Term variables		$\in$	x, y, z, f, g, h
Type variables		$\in$	a, b, c
Data constructors		$\in$	K
	ν	$::=$	$K \mid x$
Programs	$prog$	$::=$	$\epsilon \mid f = e, prog \mid f :: \sigma = e, prog$
Expressions	e	$::=$	$\nu \mid \lambda x \,.\, e \mid e_1 \; e_2 \mid \texttt{case}\ e\ \texttt{of}\{\overline{K\,\overline{x} \to e}\}$
Type schemes	σ	$::=$	$\forall \overline{a} \,.\, Q \Rightarrow \tau$
Constraints	Q	$::=$	$\epsilon \mid Q_1 \wedge Q_2 \mid \tau_1 \sim \tau_2 \mid \ldots$
Monotypes	τ, υ	$::=$	$tv \mid \texttt{Int} \mid \texttt{Bool} \mid [\tau] \mid \texttt{T}\ \overline{\tau} \mid \ldots$
	tv	$::=$	a
Type environments	Γ	$::=$	$\epsilon \mid (\nu{:}\sigma), \Gamma$
Free type variables	$ftv(\cdot)$		

Γ_0 : Types of vanilla data constructors

$K \quad : \quad \forall \overline{a} \,.\, \overline{\upsilon} \to \texttt{T}\ \overline{a}$

Top-level axiom schemes

$\mathcal{Q} \quad ::= \quad \epsilon \mid \mathcal{Q} \wedge \mathcal{Q} \mid \forall \overline{a} \,.\, Q \Rightarrow Q$

Term variables		$\in$	x, y, z, f, g, h
Type variables		$\in$	a, b, c
Data constructors		$\in$	K
	ν	$::=$	$K \mid x$
Programs	$prog$	$::=$	$\epsilon \mid f = e, prog \mid f :: \sigma = e, prog$
Expressions	e	$::=$	$\nu \mid \lambda x \,.\, e \mid e_1 \; e_2 \mid \texttt{case}\ e\ \texttt{of}\{\overline{K\,\overline{x} \to e}\}$
Type schemes	σ	$::=$	$\forall \overline{a} \,.\, Q \Rightarrow \tau$
Constraints	Q	$::=$	$\epsilon \mid Q_1 \wedge Q_2 \mid \tau_1 \sim \tau_2 \mid \ldots$
Monotypes	τ, υ	$::=$	$tv \mid \texttt{Int} \mid \texttt{Bool} \mid [\tau] \mid \texttt{T}\ \overline{\tau} \mid \ldots$
	tv	$::=$	a
Type environments	Γ	$::=$	$\epsilon \mid (\nu{:}\sigma), \Gamma$
Free type variables	$ftv(\cdot)$		

Γ_0 : Types of vanilla data constructors

$K \quad : \quad \forall \overline{a} \,.\, \overline{\upsilon} \to \texttt{T}\ \overline{a}$

$\texttt{Just} :: \forall a.a \to \text{Maybe } a$

Top-level axiom schemes

$\mathcal{Q} \quad ::= \quad \epsilon \mid \mathcal{Q} \wedge \mathcal{Q} \mid \forall \overline{a} \,.\, Q \Rightarrow Q$

Term variables		$\in$	x, y, z, f, g, h
Type variables		$\in$	a, b, c
Data constructors		$\in$	K
	ν	$::=$	$K \mid x$
Programs	$prog$	$::=$	$\epsilon \mid f = e, prog \mid f :: \sigma = e, prog$
Expressions	e	$::=$	$\nu \mid \lambda x \,.\, e \mid e_1 \; e_2 \mid \texttt{case}\ e\ \texttt{of}\{\overline{K\,\overline{x} \to e}\}$
Type schemes	σ	$::=$	$\forall \overline{a} \,.\, Q \Rightarrow \tau$
Constraints	Q	$::=$	$\epsilon \mid Q_1 \wedge Q_2 \mid \tau_1 \sim \tau_2 \mid \ldots$
Monotypes	τ, υ	$::=$	$tv \mid \texttt{Int} \mid \texttt{Bool} \mid [\tau] \mid \texttt{T}\ \overline{\tau} \mid \ldots$
	tv	$::=$	a
Type environments	Γ	$::=$	$\epsilon \mid (\nu{:}\sigma), \Gamma$
Free type variables	$ftv(\cdot)$		

Γ_0 : Types of vanilla data constructors

$K \quad : \quad \forall \overline{a} \,.\, \overline{\upsilon} \to \texttt{T}\ \overline{a}$

$\texttt{Just} :: \forall a.a \to \text{Maybe } a$

Top-level axiom schemes

$\mathcal{Q} \quad ::= \quad \epsilon \mid \mathcal{Q} \wedge \mathcal{Q} \mid \forall \overline{a} \,.\, Q \Rightarrow Q$

$\forall a.\text{Eq } a \Rightarrow \text{Eq } [a]$

Term variables		$\in$	x, y, z, f, g, h
Type variables		$\in$	a, b, c
Data constructors		$\in$	K
	ν	$::=$	$K \mid x$
Programs	$prog$	$::=$	$\epsilon \mid f = e, prog \mid f :: \sigma = e, prog$
Expressions	e	$::=$	$\nu \mid \lambda x \,.\, e \mid e_1 \; e_2 \mid \texttt{case}\ e\ \texttt{of}\{\overline{K\,\overline{x} \rightarrow e}\}$
Type schemes	σ	$::=$	$\forall \overline{a} \,.\, Q \Rightarrow \tau$
Constraints	Q	$::=$	$\epsilon \mid Q_1 \wedge Q_2 \mid \tau_1 \sim \tau_2 \mid \ldots$
Monotypes	τ, υ	$::=$	$tv \mid \texttt{Int} \mid \texttt{Bool} \mid [\tau] \mid \texttt{T}\ \overline{\tau} \mid \ldots$
	tv	$::=$	a
Type environments	Γ	$::=$	$\epsilon \mid (\nu{:}\sigma), \Gamma$
Free type variables	$ftv(\cdot)$		

Γ_0 : Types of vanilla data constructors

$K \quad : \quad \forall \overline{a} \,.\, \overline{\upsilon} \rightarrow \texttt{T}\ \overline{a}$

$\texttt{Just} :: \forall a.a \rightarrow \text{Maybe } a$

Top-level axiom schemes

$\mathcal{Q} \quad ::= \quad \epsilon \mid \mathcal{Q} \wedge \mathcal{Q} \mid \forall \overline{a} \,.\, Q \Rightarrow Q$

$\forall a.\text{Eq } a \Rightarrow \text{Eq } [a]$, Eq Int

$$\boxed{Q\,;\Gamma \vdash e : \tau}$$

$$\frac{(\nu{:}\forall \overline{a}\,.\,Q_1 \Rightarrow \upsilon) \in \Gamma \quad Q \Vdash [\overline{a \mapsto \tau}]Q_1}{Q\,;\Gamma \vdash \nu : [\overline{a \mapsto \tau}]\upsilon}\ \textsc{VarCon} \qquad \frac{Q\,;\Gamma \vdash e : \tau_1 \quad Q \Vdash \tau_1 \sim \tau_2}{Q\,;\Gamma \vdash e : \tau_2}\ \textsc{Eq}$$

$$\frac{Q\,;\Gamma, (x{:}\tau_1) \vdash \tau_2}{Q\,;\Gamma \vdash \lambda x\,.\,e : \tau_1 \to \tau_2}\ \textsc{Abs} \qquad \frac{Q\,;\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad Q\,;\Gamma \vdash e_2 : \tau_1}{Q\,;\Gamma \vdash e_1\ e_2 : \tau_2}\ \textsc{App}$$

$$\frac{\begin{array}{l} Q\,;\Gamma \vdash e : \mathtt{T}\ \overline{\tau} \\ \text{for each branch } (K_i\ \overline{x}_i \to u_i) \text{ do} \\ \quad K_i{:}\forall \overline{a}\,.\,\overline{\upsilon}_i \to \mathtt{T}\ \overline{a} \in \Gamma \quad Q\,;\Gamma, (\overline{x_i{:}[\overline{a \mapsto \tau}]\upsilon_i}) \vdash u_i : \tau_r \end{array}}{Q\,;\Gamma \vdash \mathtt{case}\ e\ \mathtt{of}\ \{\overline{K_i\ \overline{x}_i \to u_i}\} : \tau_r}\ \textsc{Case}$$

$$\boxed{\mathcal{Q} \Vdash Q}$$

$$(a \sim \text{Bool}); (x : a,\ \text{not} : \text{Bool} \to \text{Bool}) \vdash \text{not}\ x : \text{Bool}$$

Отношение $\Vdash$ должно обладать рядом свойств

$$\mathcal{Q} \wedge Q \Vdash Q$$
$$\mathcal{Q} \wedge Q_1 \Vdash Q_2 \text{ and } \mathcal{Q} \wedge Q_2 \Vdash Q_3 \text{ implies } \mathcal{Q} \wedge Q_1 \Vdash Q_3$$
$$\mathcal{Q} \Vdash Q_2 \text{ implies } \theta\mathcal{Q} \Vdash \theta Q_2 \text{ where } \theta \text{ is a type substitution}$$

$$\mathcal{Q} \Vdash \tau \sim \tau$$
$$\mathcal{Q} \Vdash \tau_1 \sim \tau_2 \text{ implies } \mathcal{Q} \Vdash \tau_2 \sim \tau_1$$
$$\mathcal{Q} \Vdash \tau_1 \sim \tau_2 \text{ and } \mathcal{Q} \Vdash \tau_2 \sim \tau_3 \text{ implies } \mathcal{Q} \Vdash \tau_1 \sim \tau_3$$
$$\mathcal{Q} \Vdash Q_1 \text{ and } \mathcal{Q} \Vdash Q_2 \text{ implies } \mathcal{Q} \Vdash Q_1 \wedge Q_2$$
$$\mathcal{Q} \Vdash \tau_1 \sim \tau_2 \text{ implies } \mathcal{Q} \Vdash [a \mapsto \tau_1]\tau \sim [a \mapsto \tau_2]\tau$$

А вот наша цель — корректно типизированная программа

$$\boxed{\mathcal{Q}\,;\,\Gamma \vdash \mathit{prog}}$$

$$\frac{\mathit{ftv}(\Gamma, \mathcal{Q}) = \emptyset}{\mathcal{Q}\,;\,\Gamma \vdash \epsilon}\ \textsc{Empty} \qquad \frac{Q_1\,;\,\Gamma \vdash e : \tau \quad \overline{a} = \mathit{ftv}(Q, \tau) \quad \mathcal{Q} \wedge Q \Vdash Q_1 \quad \mathcal{Q}\,;\,\Gamma, (f{:}\forall \overline{a}\,.\,Q \Rightarrow \tau) \vdash \mathit{prog}}{\mathcal{Q}\,;\,\Gamma \vdash f = e, \mathit{prog}}\ \textsc{Bind}$$

$$\frac{Q_1\,;\,\Gamma \vdash e : \tau \quad \overline{a} = \mathit{ftv}(Q, \tau) \quad \mathcal{Q} \wedge Q \Vdash Q_1 \quad \mathcal{Q}\,;\,\Gamma, (f{:}\forall \overline{a}\,.\,Q \Rightarrow \tau) \vdash \mathit{prog}}{\mathcal{Q}\,;\,\Gamma \vdash f\ ::\ (\forall \overline{a}\,.\,Q \Rightarrow \tau) = e, \mathit{prog}}\ \textsc{BindA}$$

Давайте уже что-нибудь типизируем, наконец

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Давайте уже что-нибудь типизируем, наконец

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: MkP : : $\forall ab.a \to b \to \mathrm{Pair}\ a\ b$.

Давайте уже что-нибудь типизируем, наконец

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: $\mathtt{MkP} :: \forall ab.a \to b \to \mathrm{Pair}\ a\ b$.

Создаём переменные унификации:

Давайте уже что-нибудь типизируем, наконец

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: $\mathtt{MkP} :: \forall ab.a \to b \to \mathrm{Pair}\ a\ b$.

Создаём переменные унификации:

- α — вся правая часть;

Давайте уже что-нибудь типизируем, наконец

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: $\texttt{MkP} :: \forall ab.a \to b \to \text{Pair}\ a\ b$.

Создаём переменные унификации:

- α — вся правая часть;
- β — тип x;

Давайте уже что-нибудь типизируем, наконец

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: $\texttt{MkP} :: \forall ab.a \to b \to \text{Pair } a\ b$.

Создаём переменные унификации:

- α — вся правая часть;
- β — тип x;
- γ_1, γ_2 — переменные, соответствующие a и b в вызове MkP.

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: MkP : : $\forall ab.a \to b \to \text{Pair } a\ b$.

Переменные унификации: $\alpha, \beta, \gamma_1, \gamma_2$.

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: $\mathtt{MkP} :: \forall ab.a \to b \to \mathrm{Pair}\ a\ b$.

Переменные унификации: $\alpha, \beta, \gamma_1, \gamma_2$. Порождаем по тексту ограничения (равенства):

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: MkP : : $\forall ab.a \to b \to \text{Pair } a\ b$.

Переменные унификации: $\alpha, \beta, \gamma_1, \gamma_2$. Порождаем по тексту ограничения (равенства):

- $\beta \sim \gamma_1$ — первый аргумент MkP;

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: MkP : : $\forall ab.a \to b \to \text{Pair } a\ b$.

Переменные унификации: $\alpha, \beta, \gamma_1, \gamma_2$. Порождаем по тексту ограничения (равенства):

- $\beta \sim \gamma_1$ — первый аргумент MkP;
- $\text{Bool} \sim \gamma_2$ — второй аргумент MkP;

```
data Pair :: * -> * -> * where
  MkP :: a -> b -> Pair a b

f x = MkP x True
```

Окружение: MkP : : $\forall ab.a \to b \to \text{Pair } a\ b$.

Переменные унификации: $\alpha, \beta, \gamma_1, \gamma_2$. Порождаем по тексту ограничения (равенства):

- $\beta \sim \gamma_1$ — первый аргумент MkP;
- $\text{Bool} \sim \gamma_2$ — второй аргумент MkP;
- $\alpha \sim \text{Pair } \gamma_1\ \gamma_2$ — результат MkP.

Система ограничений:

- $\beta \sim \gamma_1$;
- $\text{Bool} \sim \gamma_2$;
- $\alpha \sim \text{Pair}\ \gamma_1\ \gamma_2$.

Система ограничений:

- $\beta \sim \gamma_1$;
- $\text{Bool} \sim \gamma_2$;
- $\alpha \sim \text{Pair } \gamma_1\ \gamma_2$.

Решаем (унифицируем) систему ограничений и получаем подстановку:

$$\theta = [\alpha \mapsto \text{Pair } \beta\ \text{Bool}, \gamma_2 \mapsto \text{Bool}, \gamma_1 \mapsto \beta]$$

Алгоритм вывода типов мы уже знаем!

Для каждого объявления верхнего уровня:

1. Генерация ограничений
2. Решение ограничений

Расширяем синтаксис для описания алгоритма вывода типов

Unification variables	$\alpha, \beta, \gamma, \ldots$		
Unifiers	θ, φ	$::=$	$[\overline{\alpha \mapsto \tau}]$
Unification or rigid (skolem) variables	tv	$::=$	$\alpha \mid a$
Algorithm-generated constraints	C	$::=$	Q
Free unification variables	$fuv(\cdot)$		
Convert to Q-constraint	$\mathbf{simple}[Q]$	$=$	Q

Генерируем ограничения

$$\boxed{\Gamma \vdash\!\blacktriangleright\ e : \tau \rightsquigarrow C}$$

$$\frac{\overline{\alpha}\ \text{fresh} \quad (\nu{:}\forall \overline{a}\,.\,Q_1 \Rightarrow \tau_1) \in \Gamma}{\Gamma \vdash\!\blacktriangleright\ \nu : [\overline{a \mapsto \alpha}]\tau_1 \rightsquigarrow [\overline{a \mapsto \alpha}]Q_1}\ \textsc{VarCon}$$

$$\frac{\Gamma \vdash\!\blacktriangleright\ e_1 : \tau_1 \rightsquigarrow C_1 \quad \Gamma \vdash\!\blacktriangleright\ e_2 : \tau_2 \rightsquigarrow C_2 \quad \alpha\ \text{fresh}}{\Gamma \vdash\!\blacktriangleright\ e_1\ e_2 : \alpha \rightsquigarrow C_1 \wedge C_2 \wedge (\tau_1 \sim (\tau_2 \to \alpha))}\ \textsc{App}$$

$$\frac{\alpha\ \text{fresh} \quad \Gamma, (x{:}\alpha) \vdash\!\blacktriangleright\ e : \tau \rightsquigarrow C}{\Gamma \vdash\!\blacktriangleright\ \lambda x\,.\,e : \alpha \to \tau \rightsquigarrow C}\ \textsc{Abs}$$

$$\frac{\begin{array}{l}\Gamma \vdash\!\blacktriangleright e : \tau \rightsquigarrow C \quad \beta, \overline{\gamma} \text{ fresh} \quad C' = (\mathtt{T}\ \overline{\gamma} \sim \tau) \wedge C \\ \text{for each } K_i\ \overline{x}_i \to e_i \text{ do} \\ \quad K_i{:}\forall \overline{a}\,.\,\overline{\upsilon}_i \to \mathtt{T}\ \overline{a} \in \Gamma \quad \Gamma, (\overline{x_i{:}[\overline{a \mapsto \gamma}]\upsilon_i}) \vdash\!\blacktriangleright e_i : \tau_i \rightsquigarrow C_i \\ \quad C_i' = C_i \wedge \tau_i \sim \beta\end{array}}{\Gamma \vdash\!\blacktriangleright \mathtt{case}\ e\ \mathtt{of}\ \{\overline{K_i\ \overline{x}_i \to e_i}\} : \beta \rightsquigarrow C' \wedge (\bigwedge C_i')}\ \textsc{Case}$$

Решаем (упрощаем) ограничения

$$\mathcal{Q}\ ;\ Q_{given} \overset{simp}{\mapsto} Q_{wanted} \rightsquigarrow Q_{residual}\ ;\ \theta$$

Решаем (упрощаем) ограничения

$$\mathcal{Q}\ ;\ Q_{given} \overset{simp}{\mapsto} Q_{wanted} \rightsquigarrow Q_{residual}\ ;\ \theta$$

При наличии в $\mathcal{Q}$ схемы $\forall a.\mathrm{Eq}\ a \Rightarrow \mathrm{Eq}\ [a]$ в результате упрощения можно получить:

$$\mathcal{Q}\ ;\ \epsilon \overset{simp}{\mapsto} \mathtt{Eq}\ \alpha \wedge [\beta] \sim \alpha \rightsquigarrow \mathtt{Eq}\ \beta\ ;\ [\alpha \mapsto [\beta]]$$

Типизируем программу в целом

$$\boxed{\mathcal{Q}\,;\,\Gamma \mapsto prog}$$

$$\frac{}{\mathcal{Q}\,;\,\Gamma \mapsto \epsilon}\ \textsc{Empty} \qquad \frac{\begin{array}{c}\Gamma \mapsto e : \tau \rightsquigarrow Q_{wanted} \quad \mathcal{Q}\,;\,\epsilon \overset{simp}{\mapsto} Q_{wanted} \rightsquigarrow Q\,;\,\theta \\ \overline{a}\ \text{fresh} \quad \overline{\alpha} = fuv(\theta\tau, Q) \\ \mathcal{Q}\,;\,\Gamma, (f{:}\forall\overline{a}\,.\,[\overline{\alpha \mapsto a}](Q \Rightarrow \theta\tau)) \mapsto prog\end{array}}{\mathcal{Q}\,;\,\Gamma \mapsto f = e, prog}\ \textsc{Bind}$$

$$\frac{\begin{array}{c}\Gamma \mapsto e : \upsilon \rightsquigarrow Q_{wanted} \\ \mathcal{Q}\,;\,Q \overset{simp}{\mapsto} Q_{wanted} \wedge \upsilon \sim \tau \rightsquigarrow \epsilon\,;\,\theta \\ \mathcal{Q}\,;\,\Gamma, (f{:}\forall\overline{a}\,.\,Q \Rightarrow \tau) \mapsto prog\end{array}}{\mathcal{Q}\,;\,\Gamma \mapsto f\ ::\ (\forall\overline{a}\,.\,Q \Rightarrow \tau) = e, prog}\ \textsc{BindA}$$

$$\boxed{\mathcal{Q}\,;\,Q_{given} \overset{simp}{\mapsto} Q_{wanted} \rightsquigarrow Q_{residual}\,;\,\theta}$$

Всегда можно сказать, что мы программисты, и пропустить…

Всегда можно сказать, что мы программисты, и пропустить…

- Soundness

Всегда можно сказать, что мы программисты, и пропустить…

- Soundness
- Guess-free solutions

Всегда можно сказать, что мы программисты, и пропустить…

- Soundness
- Guess-free solutions
- Principality

Основанная на ограничениях система типов с локальными предположениями

$$\text{Expressions} \quad e \quad ::= \quad \ldots$$
$$\mid \quad \texttt{let}\ x :: \sigma = e_1\ \texttt{in}\ e_2$$
$$\mid \quad \texttt{let}\ x = e_1\ \texttt{in}\ e_2$$

$\ldots$

Γ_0 : Types of data constructors

$$K \quad : \quad \forall \overline{a}\ \overline{b}\ .\ Q \Rightarrow \overline{\upsilon} \rightarrow \mathtt{T}\ \overline{a}$$

Добавились ограничения в типах конструкторов данных

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a
```

Добавились ограничения в типах конструкторов данных

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a
```

Эквивалентно

```
data T :: * -> * where
  T1 :: (a ~ Bool) => Int -> T a
  T2 :: T a
```

И экзистенциальные типы

```
data X where
  Pack :: forall b. b -> (b -> Int) -> X

fx1 (Pack x f) = f x
fx2 (Pack x f) = x -- ОШИБКА
```

```haskell
data Showable where
 MkShowable :: (Show a) => a -> Showable

display :: Showable -> String
display (MkShowable x) = show x ++ "\n"
```

```
data Set :: * -> * where
  MkSet :: (Ord a) => [a] -> Set a
```

```
data Set :: * -> * where
  MkSet :: (Ord a) => [a] -> Set a

union :: Set a -> Set a -> Set a
union (MkSet xs1) (MkSet xs2)
 = MkSet (merge xs1 xs2)
-- merge :: Ord a => [a] -> [a] -> [a]
```

```haskell
data Set :: * -> * where
  MkSet :: (Ord a) => [a] -> Set a

union :: Set a -> Set a -> Set a
union (MkSet xs1) (MkSet xs2)
 = MkSet (merge xs1 xs2)
-- merge :: Ord a => [a] -> [a] -> [a]

empty :: Ord a => Set a
empty = MkSet []
```

Всё было хорошо, пока не появился let

```
f x = let g y = (x,y)
      in (g 3, g False)
```

В системе X–M для g должен выводиться тип

$$\forall b.b \to (a, b)$$

Однако...

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a
```

Однако...

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a

fr :: a -> T a -> Bool
fr x y = let gr z = not x
         in case y of
              T1 _ -> gr ()
              T2 -> True
```

Вообще-то мы могли бы типизировать gr, но надо ли?

```
gr :: forall b. (a ~ Bool) => b -> Bool
```

Проблема не только с GADT

```
fs :: a -> Set a -> Bool
fs x y = let gs z = x > z
         in case y of
              MkSet vs -> gs (head vs)
```

Проблема не только с GADT

```
fs :: a -> Set a -> Bool
fs x y = let gs z = x > z
         in case y of
              MkSet vs -> gs (head vs)

gs :: (Ord a) => a -> String
```

ЧТО ДЕЛАТЬ?

Давайте заглянем внутрь процесса разработки системы типизации

Qualified types: **Yes**, Generalization: **Yes**

$$\frac{Q_1\,;\Gamma \vdash e_1 : \tau_1 \quad \overline{a} = ftv(Q_1, \tau_1) - ftv(Q, \Gamma) \qquad Q\,;\Gamma, (x{:}\forall \overline{a}\;.\; Q_1 \Rightarrow \tau_1) \vdash e_2 : \tau_2}{Q\,;\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2}\ \text{LET}$$

Qualified types: **No**, Generalization: **Yes**

$$\frac{Q\,;\Gamma \vdash e_1 : \tau_1 \quad \overline{a} = ftv(\tau_1) - ftv(Q,\Gamma) \qquad Q\,;\Gamma, (x{:}\forall \overline{a}\,.\,\tau_1) \vdash e_2 : \tau_2}{Q\,;\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2}\ \text{LET}$$

Давайте заглянем внутрь процесса разработки системы типизации

Qualified types: **Restricted**, Generalization: **Yes**

$$\frac{Q, Q_1 \,;\Gamma \vdash e_1 : \tau_1 \quad \overline{a} = ftv(Q_1, \tau_1) - ftv(Q, \Gamma) \quad good(Q_1) \quad Q \,;\Gamma, (x{:}\forall \overline{a} \,.\, Q_1 \Rightarrow \tau_1) \vdash e_2 : \tau_2}{Q \,;\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2}\ \text{LET}$$

Давайте заглянем внутрь процесса разработки системы типизации

Qualified types: **No**, Generalization: **No**

$$\frac{Q\,;\Gamma \vdash e_1 : \tau_1 \qquad Q\,;\Gamma, (x{:}\tau_1) \vdash e_2 : \tau_2}{Q\,;\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2}\ \text{LET}$$

НЕ ПОЛУЧАЕТСЯ?

НЕ ОЧЕНЬ-ТО И ХОТЕЛОСЬ!

Окончательно приходим к следующей системе типизации

$$\boxed{Q\,;\Gamma \vdash e : \tau}$$

$$\frac{(\nu{:}\forall \overline{a}\,.\,Q_1 \Rightarrow \upsilon) \in \Gamma \quad Q \Vdash [\overline{a \mapsto \tau}]\,Q_1}{Q\,;\Gamma \vdash \nu : [\overline{a \mapsto \tau}]\upsilon}\ \textsc{VarCon} \qquad \frac{Q\,;\Gamma \vdash e : \tau_1 \quad Q \Vdash \tau_1 \sim \tau_2}{Q\,;\Gamma \vdash e : \tau_2}\ \textsc{Eq}$$

$$\frac{Q\,;\Gamma, (x{:}\tau_1) \vdash \tau_2}{Q\,;\Gamma \vdash \lambda x\,.\,e : \tau_1 \to \tau_2}\ \textsc{Abs} \qquad \frac{Q\,;\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad Q\,;\Gamma \vdash e_2 : \tau_1}{Q\,;\Gamma \vdash e_1\ e_2 : \tau_2}\ \textsc{App}$$

$$\frac{Q\,;\Gamma \vdash e_1 : \tau_1 \quad Q\,;\Gamma, (x{:}\tau_1) \vdash e_2 : \tau_2}{Q\,;\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2}\ \textsc{Let}$$

$$\frac{Q \wedge Q_1\,;\Gamma \vdash e_1 : \tau_1 \quad \overline{a} \# ftv(Q, \Gamma) \quad Q\,;\Gamma, (x{:}\forall \overline{a}\,.\,Q_1 \Rightarrow \tau_1) \vdash e_2 : \tau_2}{Q\,;\Gamma \vdash \texttt{let}\ x :: \forall \overline{a}\,.\,Q_1 \Rightarrow \tau_1 = e_1\ \texttt{in}\ e_2 : \tau_2}\ \textsc{LetA}$$

$Q\,;\Gamma \vdash e : \mathtt{T}\,\overline{\tau}$
for each branch $(K_i\ \overline{x}_i \to u_i)$ do
$K_i{:}\forall \overline{a}\ \overline{b}\ .\ Q_i \Rightarrow \overline{\upsilon}_i \to \mathtt{T}\ \overline{a} \in \Gamma$
$ftv(Q,\Gamma,\overline{\tau},\tau_r)\#\overline{b} \qquad Q \wedge ([\overline{a \mapsto \tau}]Q_i)\ ;\Gamma, (\overline{x_i{:}[\overline{a \mapsto \tau}]\upsilon_i}) \vdash u_i : \tau_r$

$$\frac{}{Q\,;\Gamma \vdash \mathtt{case}\ e\ \mathtt{of}\ \{\overline{K_i\ \overline{x}_i \to u_i}\} : \tau_r}\ \textsc{Case}$$

$$\boxed{\mathcal{Q}\,;\Gamma \vdash \mathit{prog}}$$

$$\frac{\mathit{ftv}(\Gamma,\mathcal{Q}) = \emptyset}{\mathcal{Q}\,;\Gamma \vdash \epsilon}\ \textsc{Empty} \qquad \frac{Q_1\,;\Gamma \vdash e : \tau \quad \overline{a} = \mathit{ftv}(Q,\tau) \quad \mathcal{Q} \wedge Q \Vdash Q_1 \quad \mathcal{Q}\,;\Gamma,(f{:}\forall \overline{a}\,.\,Q \Rightarrow \tau) \vdash \mathit{prog}}{\mathcal{Q}\,;\Gamma \vdash f = e, \mathit{prog}}\ \textsc{Bind}$$

$$\frac{Q_1\,;\Gamma \vdash e : \tau \quad \overline{a} = \mathit{ftv}(Q,\tau) \quad \mathcal{Q} \wedge Q \Vdash Q_1 \quad \mathcal{Q}\,;\Gamma,(f{:}\forall \overline{a}\,.\,Q \Rightarrow \tau) \vdash \mathit{prog}}{\mathcal{Q}\,;\Gamma \vdash f :: \forall \overline{a}\,.\,Q \Rightarrow \tau = e, \mathit{prog}}\ \textsc{BindA}$$

$$\boxed{\mathcal{Q} \Vdash Q}$$

Помните этот пример?

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a

test (T1 n) _ = n > 0
test T2 r = r
```

$\texttt{test} :: \forall a.\mathrm{T}\ a \to \mathrm{Bool} \to \mathrm{Bool}$

$\texttt{test} :: \forall a.\mathrm{T}\ a \to a \to a$

Помните этот пример?

```
data T :: * -> * where
  T1 :: Int -> T Bool
  T2 :: T a

test (T1 n) _ = n > 0
test T2 r = r
```

$$\texttt{test} :: \forall a.\, \mathrm{T}\ a \to \mathrm{Bool} \to \mathrm{Bool}$$

$$\texttt{test} :: \forall a.\, \mathrm{T}\ a \to a \to a$$

Эта система типов его допускает!

НУ И ПУСТЬ!

ЛИШЬ БЫ ТИП НЕ ВЫВОДИЛСЯ…

Вывод типов в системе OutsideIn(X)

Начнём с примеров

$$\mathtt{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$$

```
\x -> case x of { T1 n -> n > 0 }
```

Начнём с примеров

$\mathtt{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$

```
\x -> case x of { T1 n -> n > 0 }
```

- Создаём переменные α для всего тела и β_x для x

Начнём с примеров

$\mathrm{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$

```
\x -> case x of { T1 n -> n > 0 }
```

- Создаём переменные α для всего тела и β_x для x
- $\beta_x \sim \mathrm{T}\ \gamma$ (для свежей γ)

Начнём с примеров

$\mathrm{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$

```
\x -> case x of { T1 n -> n > 0 }
```

- Создаём переменные α для всего тела и β_x для x
- $\beta_x \sim \mathrm{T}\ \gamma$ (для свежей γ)
- $\alpha \sim \mathrm{Bool}$ (но в той же ветке возникает $\gamma \sim \mathrm{Bool}$)

Начнём с примеров

$\mathrm{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$

```
\x -> case x of { T1 n -> n > 0 }
```

- Создаём переменные α для всего тела и β_x для x
- $\beta_x \sim \mathrm{T}\ \gamma$ (для свежей γ)
- $\alpha \sim \mathrm{Bool}$ (но в той же ветке возникает $\gamma \sim \mathrm{Bool}$)
- $\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$ (ограничение-ипликация)

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$$

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$$

Есть два унификатора:

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$$

Есть два унификатора: $[\alpha \mapsto \mathrm{Bool}]$ и $[\alpha \mapsto \gamma]$.

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$$

Есть два унификатора: $[\alpha \mapsto \mathrm{Bool}]$ и $[\alpha \mapsto \gamma]$.

А вот здесь всё ясно (`test2`):

```
\x -> case x of
        T1 n -> n > 0
        T2 -> True
```

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}$$

Есть два унификатора: $[\alpha \mapsto \text{Bool}]$ и $[\alpha \mapsto \gamma]$.

А вот здесь всё ясно (`test2`):

```
\x -> case x of
        T1 n -> n > 0
        T2 -> True
```

$$(\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}) \wedge (\alpha \sim \text{Bool})$$

Мы не унифицируем “глобальные” переменные в импликациях под равенством

$$\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$$

Мы не унифицируем “глобальные” переменные в импликациях под равенством

$\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}$

Но сравните:

$\texttt{T3} :: \forall a.(\text{Bool} \sim a) \Rightarrow [\text{Int}] \to \text{T}\ a$

$\texttt{null} :: \forall d.[\text{d}] \to \text{Bool}\ a$

```
\x -> case x of { T3 ns -> null ns }
```

Мы не унифицируем “глобальные” переменные в импликациях под равенством

$\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}$

Но сравните:

$\texttt{T3} :: \forall a.(\text{Bool} \sim a) \Rightarrow [\text{Int}] \to \text{T}\ a$

$\texttt{null} :: \forall d.[\text{d}] \to \text{Bool}\ a$

```
\x -> case x of { T3 ns -> null ns }
```

В момент работы с `null ns` мы получаем $\delta \sim \text{Int}$:

$\gamma \sim \text{Bool} \supset (\alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$

Есть разница: α и γ глобальные, δ локальная

$$\gamma \sim \text{Bool} \supset (\alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

Есть разница: α и γ глобальные, δ локальная

$$\gamma \sim \text{Bool} \supset (\alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

превращается в

$$\exists \delta.(\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

(новый вид ограничений)

Есть разница: α и γ глобальные, δ локальная

$$\gamma \sim \text{Bool} \supset (\alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

превращается в

$$\exists \delta.(\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

(новый вид ограничений)

Локальные для импликаций ограничения можно унифицировать!

Расширяем синтаксис для OutsideIn(X)

Unification variables	$\alpha, \beta, \gamma, \ldots$		
Unification or skolem variables	tv	$::=$	$\alpha \mid a$
Algorithm-generated constraints	C	$::=$	$Q \mid C_1 \wedge C_2 \mid \exists \overline{\alpha}.(Q \supset C)$
Free unification variables	$fuv(\cdot)$		

$$
\begin{array}{lcl}
\mathbf{simple}[Q] & = & Q \\
\mathbf{simple}[C_1 \wedge C_2] & = & \mathbf{simple}[C_1] \wedge \mathbf{simple}[C_2] \\
\mathbf{simple}[\exists \overline{\alpha}.(Q \supset C)] & = & \epsilon \\
 & & \\
\mathbf{implic}[Q] & = & \epsilon \\
\mathbf{implic}[C_1 \wedge C_2] & = & \mathbf{implic}[C_1] \wedge \mathbf{implic}[C_2] \\
\mathbf{implic}[\exists \overline{\alpha}.(Q \supset C)] & = & \exists \overline{\alpha}.(Q \supset C)
\end{array}
$$

$\overline{\alpha}$ — *прикасаемые переменные (touchables)*

Как теперь решать ограничения?

1. Делим ограничения на простые и импликации
2. Решаем простые солвером для X
3. Используя результат (2) решаем импликации (унифицируя только прикасаемые переменные)

Приходим к общей схеме алгоритма OutsideIn(X)

$$\boxed{\mathcal{Q}\,;\,\Gamma \mapsto prog}$$

$$\frac{}{\mathcal{Q}\,;\,\Gamma \mapsto \epsilon}\ \textsc{Empty} \qquad \frac{\Gamma \mapsto e : \tau \rightsquigarrow C \qquad \mathcal{Q}\,;\,\epsilon\,;\,fuv(\tau, C) \overset{solv}{\mapsto} C \rightsquigarrow Q\,;\,\theta \qquad \overline{a}\ \text{fresh} \quad \overline{\alpha} = fuv(\theta\tau, Q) \qquad \mathcal{Q}\,;\,\Gamma, (f{:}\forall \overline{a}\,.\,[\overline{\alpha \mapsto a}](Q \Rightarrow \theta\tau)) \mapsto prog}{\mathcal{Q}\,;\,\Gamma \mapsto f = e, prog}\ \textsc{Bind}$$

$$\frac{\Gamma \mapsto e : \upsilon \rightsquigarrow C \qquad \mathcal{Q}\,;\,Q\,;\,fuv(\upsilon, C) \overset{solv}{\mapsto} C \wedge \upsilon \sim \tau \rightsquigarrow \epsilon\,;\,\theta \qquad \mathcal{Q}\,;\,\Gamma, (f{:}\forall \overline{a}\,.\,Q \Rightarrow \tau) \mapsto prog}{\mathcal{Q}\,;\,\Gamma \mapsto f\ ::\ (\forall \overline{a}\,.\,Q \Rightarrow \tau) = e, prog}\ \textsc{BindA}$$

$$\boxed{\mathcal{Q}\,;\,Q_{given}\,;\,\overline{\alpha}_{tch} \overset{solv}{\mapsto} C_{wanted} \rightsquigarrow Q_{residual}\,;\,\theta}$$

Дополняем генерацию ограничений

$$\boxed{\Gamma \mapsto e : \tau \rightsquigarrow C}$$

$$\ldots$$

$$\frac{\Gamma \mapsto e_1 : \tau_1 \rightsquigarrow C_1 \quad \Gamma, (x{:}\tau_1) \mapsto e_2 : \tau_2 \rightsquigarrow C_2}{\Gamma \mapsto \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2 \rightsquigarrow C_1 \wedge C_2}\ \textsc{Let}$$

$$\frac{\Gamma \mapsto e_1 : \tau \rightsquigarrow C_1 \quad \Gamma, (x{:}\tau_1) \mapsto e_2 : \tau_2 \rightsquigarrow C_2}{\Gamma \mapsto \texttt{let}\ x :: \tau_1 = e_1\ \texttt{in}\ e_2 : \tau_2 \rightsquigarrow C_1 \wedge C_2 \wedge \tau \sim \tau_1}\ \textsc{LetA}$$

$$\frac{\begin{array}{c}\sigma_1 = \forall \overline{a}\,.\, Q_1 \Rightarrow \tau_1 \quad Q_1 \neq \epsilon \text{ or } \overline{a} \neq \epsilon \quad \Gamma \mapsto e_1 : \tau \rightsquigarrow C \quad \overline{\beta} = \mathit{fuv}(\tau, C) - \mathit{fuv}(\Gamma) \\ C_1 = \exists \overline{\beta}\,.\, (Q_1 \supset C \wedge \tau \sim \tau_1) \quad \Gamma, (x{:}\sigma_1) \mapsto e_2 : \tau_2 \rightsquigarrow C_2\end{array}}{\Gamma \mapsto \texttt{let}\ x ::\sigma_1 = e_1\ \texttt{in}\ e_2 : \tau_2 \rightsquigarrow C_1 \wedge C_2}\ \textsc{GLetA}$$

$$\frac{\begin{array}{c}\Gamma \vdash\!\blacktriangleright e : \tau \rightsquigarrow C \quad \beta, \overline{\gamma} \text{ fresh} \\ K_i{:}\forall \overline{a}\,\overline{b}_i \,.\, Q_i \Rightarrow \overline{\upsilon}_i \to \mathtt{T}\ \overline{a} \quad \overline{b}_i \text{ fresh} \\ \Gamma, (\overline{x_i{:}[\overline{a \mapsto \gamma}]\upsilon_i}) \vdash\!\blacktriangleright e_i : \tau_i \rightsquigarrow C_i \quad \overline{\delta}_i = fuv(\tau_i, C_i) - fuv(\Gamma, \overline{\gamma}) \\ C'_i = \begin{cases} C_i \wedge \tau_i \sim \beta & \text{if } \overline{b}_i = \epsilon \text{ and } Q_i = \epsilon \\ \exists \overline{\delta}_i \,.\, ([\overline{a \mapsto \gamma}]\, Q_i \supset C_i \wedge \tau_i \sim \beta) & \text{otherwise} \end{cases}\end{array}}{\Gamma \vdash\!\blacktriangleright \mathtt{case}\ e\ \mathtt{of}\ \{\overline{K_i\ \overline{x}_i \to e_i}\} : \beta \rightsquigarrow C \wedge (\mathtt{T}\ \overline{\gamma} \sim \tau) \wedge (\bigwedge C'_i)}\ \textsc{Case}$$

Создаём инфраструктуру решателя ограничений

$$\boxed{\mathcal{Q}\,;\, Q_{given}\,;\, \overline{\alpha}_{tch} \overset{solv}{\mapsto} C_{wanted} \rightsquigarrow Q_{residual}\,;\, \theta}$$

$$\frac{\begin{array}{l}\mathcal{Q}\,;\, Q_g\,;\, \overline{\alpha} \overset{simp}{\mapsto} \mathbf{simple}[C] \rightsquigarrow Q_r\,;\, \theta \\ \forall(\exists \overline{\alpha}_i\,.\,(Q_i \supset C_i) \in \mathbf{implic}[\theta\, C]), \\ \quad \mathcal{Q}\,;\, Q_g \wedge Q_r \wedge Q_i\,;\, \overline{\alpha}_i \overset{solv}{\mapsto} C_i \rightsquigarrow \epsilon\,;\, \theta_i\end{array}}{\mathcal{Q}\,;\, Q_g\,;\, \overline{\alpha} \overset{solv}{\mapsto} C \rightsquigarrow Q_r\,;\, \theta}\ \text{SOLVE}$$

$$\boxed{\mathcal{Q}\,;\, Q_{given}\,;\, \overline{\alpha}_{tch} \overset{simp}{\mapsto} Q_{wanted} \rightsquigarrow Q_{residual}\,;\, \theta}$$

И формулируем основную теорему

И формулируем основную теорему

Теорема. Если решатель (simplifier) для X хороший, то всё хорошо.

И формулируем основную теорему

Теорема. Если решатель (simplifier) для X хороший, то всё хорошо.

Доказательство. С очевидностью следует из теорем 5.1 и 5.2 (Vytiniotis и др., 2011).

Мы, правда, чисто по программистски сделали то, что смогли

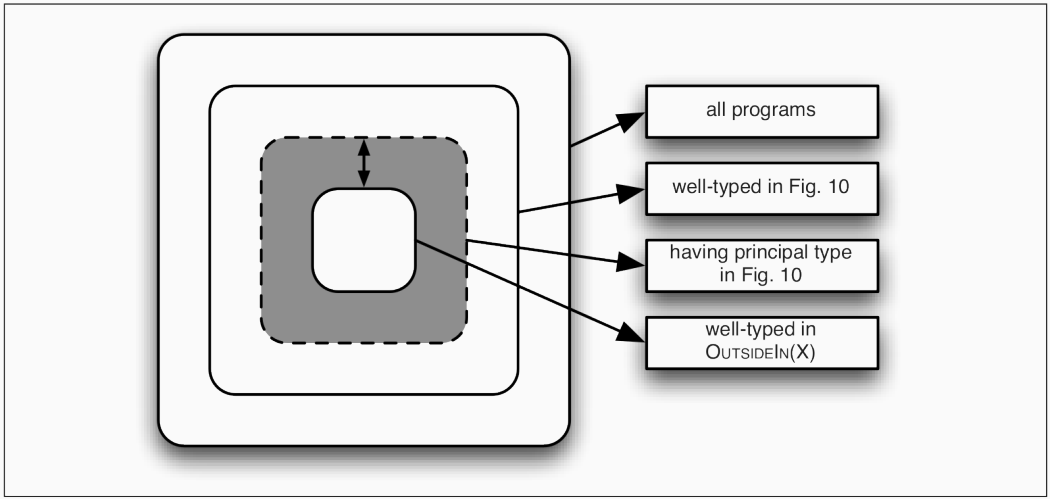


Fig. 16: The space of programs

Вывод типов в системе OutsideIn(X)

Начнём с примеров

$$\mathtt{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$$

```
\x -> case x of { T1 n -> n > 0 }
```

Начнём с примеров

$\mathtt{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$

```
\x -> case x of { T1 n -> n > 0 }
```

- Создаём переменные α для всего тела и β_x для x

Начнём с примеров

$\mathrm{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$

```
\x -> case x of { T1 n -> n > 0 }
```

- Создаём переменные α для всего тела и β_x для x
- $\beta_x \sim \mathrm{T}\ \gamma$ (для свежей γ)

Начнём с примеров

$\mathrm{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$

```
\x -> case x of { T1 n -> n > 0 }
```

- Создаём переменные α для всего тела и β_x для x
- $\beta_x \sim \mathrm{T}\ \gamma$ (для свежей γ)
- $\alpha \sim \mathrm{Bool}$ (но в той же ветке возникает $\gamma \sim \mathrm{Bool}$)

Начнём с примеров

$\mathrm{T1} :: \forall a.(\mathrm{Bool} \sim a) \Rightarrow \mathrm{Int} \to \mathrm{T}\ a$

```
\x -> case x of { T1 n -> n > 0 }
```

- Создаём переменные α для всего тела и β_x для x
- $\beta_x \sim \mathrm{T}\ \gamma$ (для свежей γ)
- $\alpha \sim \mathrm{Bool}$ (но в той же ветке возникает $\gamma \sim \mathrm{Bool}$)
- $\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$ (ограничение-ипликация)

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$$

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$$

Есть два унификатора:

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \mathrm{Bool} \supset \alpha \sim \mathrm{Bool}$$

Есть два унификатора: $[\alpha \mapsto \mathrm{Bool}]$ и $[\alpha \mapsto \gamma]$.

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}$$

Есть два унификатора: $[\alpha \mapsto \text{Bool}]$ и $[\alpha \mapsto \gamma]$.

А вот здесь всё ясно (`test2`):

```
\x -> case x of
        T1 n -> n > 0
        T2 -> True
```

В этот момент мы ясно видим проблему — выявляем неоднозначность

$$\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}$$

Есть два унификатора: $[\alpha \mapsto \text{Bool}]$ и $[\alpha \mapsto \gamma]$.

А вот здесь всё ясно (`test2`):

```
\x -> case x of
        T1 n -> n > 0
        T2 -> True
```

$$(\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}) \wedge (\alpha \sim \text{Bool})$$

Мы не унифицируем “глобальные” переменные в импликациях под равенством

$$\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}$$

Мы не унифицируем “глобальные” переменные в импликациях под равенством

$\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}$

Но сравните:

$\texttt{T3} :: \forall a.(\text{Bool} \sim a) \Rightarrow [\text{Int}] \rightarrow \text{T}\ a$

$\texttt{null} :: \forall d.[\text{d}] \rightarrow \text{Bool}\ a$

```
\x -> case x of { T3 ns -> null ns }
```

Мы не унифицируем “глобальные” переменные в импликациях под равенством

$\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool}$

Но сравните:

$\texttt{T3} :: \forall a.(\text{Bool} \sim a) \Rightarrow [\text{Int}] \to \text{T}\ a$

$\texttt{null} :: \forall d.[\text{d}] \to \text{Bool}\ a$

```
\x -> case x of { T3 ns -> null ns }
```

В момент работы с `null ns` мы получаем $\delta \sim \text{Int}$:

$\gamma \sim \text{Bool} \supset (\alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$

Есть разница: α и γ глобальные, δ локальная

$$\gamma \sim \text{Bool} \supset (\alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

Есть разница: α и γ глобальные, δ локальная

$$\gamma \sim \text{Bool} \supset (\alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

превращается в

$$\exists \delta.(\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

(новый вид ограничений)

Есть разница: α и γ глобальные, δ локальная

$$\gamma \sim \text{Bool} \supset (\alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

превращается в

$$\exists \delta.(\gamma \sim \text{Bool} \supset \alpha \sim \text{Bool} \wedge \delta \sim \text{Int})$$

(новый вид ограничений)

Локальные для импликаций ограничения можно унифицировать!

Расширяем синтаксис для OutsideIn(X)

Unification variables	$\alpha, \beta, \gamma, \ldots$		
Unification or skolem variables	tv	$::=$	$\alpha \mid a$
Algorithm-generated constraints	C	$::=$	$Q \mid C_1 \wedge C_2 \mid \exists \overline{\alpha}.(Q \supset C)$
Free unification variables	$fuv(\cdot)$		

$$
\begin{array}{lcl}
\mathbf{simple}[Q] & = & Q \\
\mathbf{simple}[C_1 \wedge C_2] & = & \mathbf{simple}[C_1] \wedge \mathbf{simple}[C_2] \\
\mathbf{simple}[\exists \overline{\alpha}.(Q \supset C)] & = & \epsilon \\
 & & \\
\mathbf{implic}[Q] & = & \epsilon \\
\mathbf{implic}[C_1 \wedge C_2] & = & \mathbf{implic}[C_1] \wedge \mathbf{implic}[C_2] \\
\mathbf{implic}[\exists \overline{\alpha}.(Q \supset C)] & = & \exists \overline{\alpha}.(Q \supset C)
\end{array}
$$

$\overline{\alpha}$ — *прикасаемые переменные (touchables)*

Как теперь решать ограничения?

1. Делим ограничения на простые и импликации
2. Решаем простые солвером для X
3. Используя результат (2) решаем импликации (унифицируя только прикасаемые переменные)

Приходим к общей схеме алгоритма OutsideIn(X)

$$\boxed{\mathcal{Q} \,;\, \Gamma \vdash\!\blacktriangleright\ prog}$$

$$\frac{}{\mathcal{Q} \,;\, \Gamma \vdash\!\blacktriangleright\ \epsilon}\ \textsc{Empty} \qquad \frac{\Gamma \vdash\!\blacktriangleright\ e : \tau \rightsquigarrow C \qquad \mathcal{Q} \,;\, \epsilon \,;\, fuv(\tau, C) \overset{solv}{\vdash\!\blacktriangleright} C \rightsquigarrow Q \,;\, \theta \quad \overline{a}\ \text{fresh} \quad \overline{\alpha} = fuv(\theta\tau, Q) \qquad \mathcal{Q} \,;\, \Gamma, (f{:}\forall \overline{a} \,.\, [\overline{\alpha \mapsto a}](Q \Rightarrow \theta\tau)) \vdash\!\blacktriangleright\ prog}{\mathcal{Q} \,;\, \Gamma \vdash\!\blacktriangleright\ f = e, prog}\ \textsc{Bind}$$

$$\frac{\Gamma \vdash\!\blacktriangleright\ e : \upsilon \rightsquigarrow C \qquad \mathcal{Q} \,;\, Q \,;\, fuv(\upsilon, C) \overset{solv}{\vdash\!\blacktriangleright} C \wedge \upsilon \sim \tau \rightsquigarrow \epsilon \,;\, \theta \qquad \mathcal{Q} \,;\, \Gamma, (f{:}\forall \overline{a} \,.\, Q \Rightarrow \tau) \vdash\!\blacktriangleright\ prog}{\mathcal{Q} \,;\, \Gamma \vdash\!\blacktriangleright\ f\ ::\ (\forall \overline{a} \,.\, Q \Rightarrow \tau) = e, prog}\ \textsc{BindA}$$

$$\boxed{\mathcal{Q} \,;\, Q_{given} \,;\, \overline{\alpha}_{tch} \overset{solv}{\vdash\!\blacktriangleright} C_{wanted} \rightsquigarrow Q_{residual} \,;\, \theta}$$

Дополняем генерацию ограничений

$$\boxed{\Gamma \mapsto e : \tau \rightsquigarrow C}$$

$$\ldots$$

$$\frac{\Gamma \mapsto e_1 : \tau_1 \rightsquigarrow C_1 \quad \Gamma, (x{:}\tau_1) \mapsto e_2 : \tau_2 \rightsquigarrow C_2}{\Gamma \mapsto \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau_2 \rightsquigarrow C_1 \wedge C_2}\ \textsc{Let}$$

$$\frac{\Gamma \mapsto e_1 : \tau \rightsquigarrow C_1 \quad \Gamma, (x{:}\tau_1) \mapsto e_2 : \tau_2 \rightsquigarrow C_2}{\Gamma \mapsto \texttt{let}\ x :: \tau_1 = e_1\ \texttt{in}\ e_2 : \tau_2 \rightsquigarrow C_1 \wedge C_2 \wedge \tau \sim \tau_1}\ \textsc{LetA}$$

$$\frac{\begin{array}{c}\sigma_1 = \forall \overline{a}\,.\, Q_1 \Rightarrow \tau_1 \quad Q_1 \neq \epsilon \text{ or } \overline{a} \neq \epsilon \quad \Gamma \mapsto e_1 : \tau \rightsquigarrow C \quad \overline{\beta} = \mathit{fuv}(\tau, C) - \mathit{fuv}(\Gamma) \\ C_1 = \exists \overline{\beta}\,.\, (Q_1 \supset C \wedge \tau \sim \tau_1) \quad \Gamma, (x{:}\sigma_1) \mapsto e_2 : \tau_2 \rightsquigarrow C_2\end{array}}{\Gamma \mapsto \texttt{let}\ x :: \sigma_1 = e_1\ \texttt{in}\ e_2 : \tau_2 \rightsquigarrow C_1 \wedge C_2}\ \textsc{GLetA}$$

$$\Gamma \vdash\!\blacktriangleright e : \tau \rightsquigarrow C \quad \beta, \overline{\gamma} \text{ fresh}$$

$$K_i{:}\forall \overline{a}\,\overline{b}_i \,.\, Q_i \Rightarrow \overline{\upsilon}_i \to \mathtt{T}\ \overline{a} \quad \overline{b}_i \text{ fresh}$$

$$\Gamma, (\overline{x_i{:}[\overline{a \mapsto \gamma}]\upsilon_i}) \vdash\!\blacktriangleright e_i : \tau_i \rightsquigarrow C_i \quad \overline{\delta}_i = \mathit{fuv}(\tau_i, C_i) - \mathit{fuv}(\Gamma, \overline{\gamma})$$

$$C'_i = \begin{cases} C_i \wedge \tau_i \sim \beta & \text{if } \overline{b}_i = \epsilon \text{ and } Q_i = \epsilon \\ \exists \overline{\delta}_i \,.\, ([\overline{a \mapsto \gamma}]\, Q_i \supset C_i \wedge \tau_i \sim \beta) & \text{otherwise} \end{cases}$$

$$\Gamma \vdash\!\blacktriangleright \mathtt{case}\ e\ \mathtt{of}\ \{\overline{K_i\ \overline{x}_i \to e_i}\} : \beta \rightsquigarrow C \wedge (\mathtt{T}\ \overline{\gamma} \sim \tau) \wedge (\bigwedge C'_i) \qquad \textsc{Case}$$

Создаём инфраструктуру решателя ограничений

$$\boxed{\mathcal{Q}\,;\,Q_{given}\,;\,\overline{\alpha}_{tch} \overset{solv}{\mapsto} C_{wanted} \rightsquigarrow Q_{residual}\,;\,\theta}$$

$$\frac{\begin{array}{l}\mathcal{Q}\,;\,Q_g\,;\,\overline{\alpha} \overset{simp}{\mapsto} \mathbf{simple}[C] \rightsquigarrow Q_r\,;\,\theta \\ \forall(\exists\overline{\alpha}_i\,.\,(Q_i \supset C_i) \in \mathbf{implic}[\theta\,C]), \\ \quad \mathcal{Q}\,;\,Q_g \wedge Q_r \wedge Q_i\,;\,\overline{\alpha}_i \overset{solv}{\mapsto} C_i \rightsquigarrow \epsilon\,;\,\theta_i\end{array}}{\mathcal{Q}\,;\,Q_g\,;\,\overline{\alpha} \overset{solv}{\mapsto} C \rightsquigarrow Q_r\,;\,\theta}\ \text{SOLVE}$$

$$\boxed{\mathcal{Q}\,;\,Q_{given}\,;\,\overline{\alpha}_{tch} \overset{simp}{\mapsto} Q_{wanted} \rightsquigarrow Q_{residual}\,;\,\theta}$$

И формулируем основную теорему

И формулируем основную теорему

Теорема. Если решатель (simplifier) для X хороший, то всё хорошо.

И формулируем основную теорему

Теорема. Если решатель (simplifier) для X хороший, то всё хорошо.

Доказательство. С очевидностью следует из теорем 5.1 и 5.2 (Vytiniotis и др., 2011).

Мы, правда, чисто по программистски сделали то, что смогли

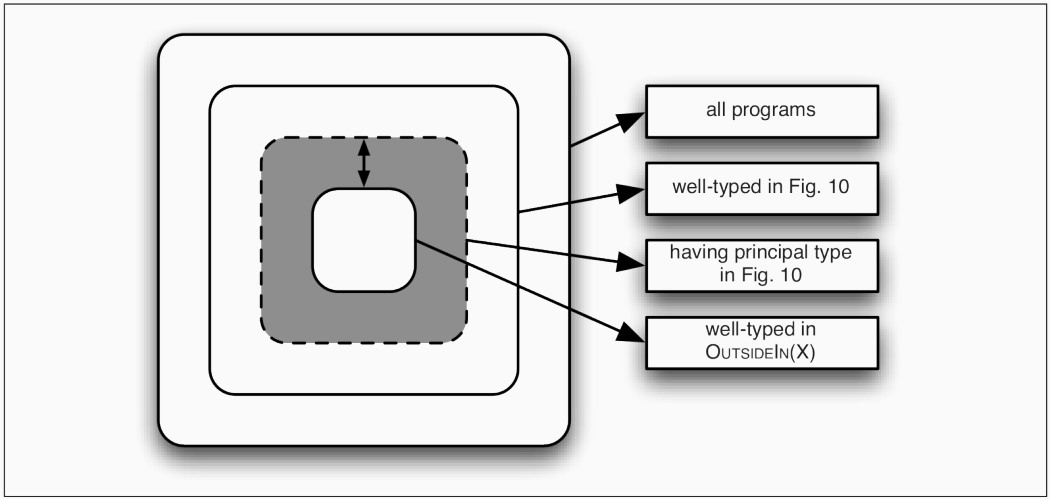


Fig. 16: The space of programs

Инстанцирование X для GADT, классов типов и семейств типов

Расширяем синтаксис

Syntactic extensions

$$\begin{array}{lcll} \tau & ::= & \ldots \mid F\ \overline{\tau} & \text{Type family applications} \\ Q & ::= & \ldots \mid \mathtt{D}\ \overline{\tau} & \text{Type class constraints} \\ \mathcal{Q} & ::= & Q \mid \mathcal{Q} \wedge \mathcal{Q} \mid \forall \overline{a}\,.\,Q \Rightarrow \mathtt{D}\ \overline{\tau} \mid \forall \overline{a}\,.\,F\ \overline{\xi} \sim \tau & \end{array}$$

Auxiliary syntactic definitions

$$\begin{array}{lcl} \zeta, \xi & \in & \{\tau \mid \tau \text{ contains no type families } \} \\ \mathbb{T} & ::= & \mathtt{T}\ \overline{\mathbb{T}} \mid \mathbb{F} \mid \mathbb{T} \to \mathbb{T} \mid tv \mid \bullet \\ \mathbb{F} & ::= & F\ \overline{\mathbb{T}} \\ \mathbb{D} & ::= & D\ \overline{\mathbb{T}} \end{array}$$

Упрощаем ограничения

$$\boxed{\mathcal{Q}\,;\,Q_{given}\,;\,\overline{\alpha}_{tch} \overset{simp}{\mapsto\!\!\blacktriangleright} Q_{wanted} \rightsquigarrow Q_{residual}\,;\,\theta}$$

$$\frac{\begin{array}{l}\mathcal{Q} \mapsto\!\!\blacktriangleright \langle \overline{\alpha}, \epsilon, Q_g, Q_w\rangle \hookrightarrow^{\star} \langle \overline{\alpha}', \varphi, Q_g', Q_w'\rangle \not\hookrightarrow \\ \quad \varphi Q_w' = \mathcal{E} \wedge Q_r \\ \mathcal{E} = \{\beta \sim \tau \mid ((\beta \sim \tau) \in \varphi Q_w' \text{ or } (\tau \sim \beta) \in \varphi Q_w'), \beta \in \overline{\alpha}', \beta \notin fuv(\tau)\} \\ \theta = [\beta \mapsto \theta\tau \mid (\beta \sim \tau) \in \mathcal{E}] \qquad \overline{\beta} \text{ distinct}\end{array}}{\mathcal{Q}\,;\,Q_g\,;\,\overline{\alpha} \overset{simp}{\mapsto\!\!\blacktriangleright} Q_w \rightsquigarrow \theta Q_r\,;\,\theta|_{\overline{\alpha}}}\ \text{SIMPLES}$$

Переписываем ограничения

$$\boxed{\mathcal{Q} \vdash\!\blacktriangleright \langle \overline{\alpha}, \varphi, Q_g, Q_w \rangle \hookrightarrow \langle \overline{\alpha}', \varphi', Q_g', Q_w' \rangle_\perp}$$

$$\frac{canon[g]\ (Q_1) = \{\overline{\beta}, \varphi_2, Q_2\}_\perp}{\mathcal{Q} \vdash\!\blacktriangleright \langle \overline{\alpha}, \varphi_1, Q_g \wedge Q_1, Q_w \rangle \hookrightarrow \langle \overline{\alpha}\overline{\beta}, \varphi_1 \uplus \varphi_2, Q_g \wedge Q_2, Q_w \rangle_\perp}\ \text{CANG}$$

$$\frac{canon[w]\ (Q_1) = \{\overline{\beta}, \varphi_2, Q_2\}_\perp}{\mathcal{Q} \vdash\!\blacktriangleright \langle \overline{\alpha}, \varphi_1, Q_g, Q_w \wedge Q_1 \rangle \hookrightarrow \langle \overline{\alpha}\overline{\beta}, \varphi_1 \uplus \varphi_2, Q_g, Q_w \wedge Q_2 \rangle_\perp}\ \text{CANW}$$

$$\frac{interact[g](Q_1,\ Q_2) = Q_3}{\mathcal{Q} \vdash\!\blacktriangleright \langle \overline{\alpha}, \varphi, Q_g \wedge\ Q_1 \wedge Q_2, Q_w \rangle \hookrightarrow \langle \overline{\alpha}, \varphi, Q_g \wedge Q_3, Q_w \rangle_\perp}\ \text{INTG}$$

$$\frac{interact[w](Q_1,\ Q_2) = Q_3}{\mathcal{Q} \vdash\!\blacktriangleright \langle \overline{\alpha}, \varphi, Q_g, Q_w \wedge Q_1 \wedge Q_2 \rangle \hookrightarrow \langle \overline{\alpha}, \varphi, Q_g, Q_w \wedge Q_3 \rangle_\perp}\ \text{INTW}$$

$$\frac{(Q)\ \ simplifies\ (Q_1) = Q_2}{\mathcal{Q} \vdash\!\blacktriangleright \langle \overline{\alpha}, \varphi, Q_g \wedge Q, Q_w \wedge Q_1 \rangle \hookrightarrow \langle \overline{\alpha}, \varphi, Q_g \wedge Q, Q_w \wedge Q_2 \rangle_\perp}\ \text{SIMPL}$$

$$\frac{topreact[g](\mathcal{Q},\ Q_1) = \{\epsilon, Q_2\}_\perp}{\mathcal{Q} \vdash\!\blacktriangleright \langle \overline{\alpha}, \varphi, Q_g \wedge Q_1, Q_w \rangle \hookrightarrow \langle \overline{\alpha}, \varphi, Q_g \wedge Q_2, Q_w \rangle_\perp}\ \text{TOPG}$$

$$\frac{topreact[w](\mathcal{Q},\ Q_1) = \{\overline{\beta}, Q_2\}_\perp}{\mathcal{Q} \vdash\!\blacktriangleright \langle \overline{\alpha}, \varphi, Q_g, Q_w \wedge Q_1 \rangle \hookrightarrow \langle \overline{\alpha}\overline{\beta}, \varphi, Q_g, Q_w \wedge Q_2 \rangle_\perp}\ \text{TOPW}$$

- Каноникализация
- Бинарное взаимодействие
- Упрощение
- Переписывание с использованием аксиом верхнего уровня

Определяем вспомогательные функции

$$\boxed{canon[\ell]\ (Q_1) = \{\overline{\beta}, \varphi, Q_2\}_\perp}$$

REFL	$canon[\ell]\ (\tau \sim \tau)$	$=$	$\{\epsilon, \epsilon, \epsilon\}$
TDEC	$canon[\ell]\ (\mathtt{T}\ \overline{\tau}_1 \sim \mathtt{T}\ \overline{\tau}_2)$	$=$	$\{\epsilon, \epsilon, \bigwedge \overline{\tau_1 \sim \tau_2}\}$
FAILDEC	$canon[\ell]\ (\mathtt{T}\ \overline{\tau}_1 \sim \mathtt{S}\ \overline{\tau}_2)$	$=$	$\perp$
OCCCHECK	$canon[\ell]\ (tv \sim \xi)$		
	where $tv \in \xi, \xi \neq tv$	$=$	$\perp$
ORIENT	$canon[\ell]\ (\tau_1 \sim \tau_2)$		
	where $\tau_2 \prec \tau_1$	$=$	$\{\epsilon, \epsilon, \tau_2 \sim \tau_1\}$
DFLATW	$canon[w]\ (\mathbb{D}[G\ \overline{\xi}])$	$=$	$\{\beta, \epsilon, \mathbb{D}[\beta] \wedge (G\ \overline{\xi} \sim \beta)\}$
DFLATG	$canon[g]\ (\mathbb{D}[G\ \overline{\xi}])$	$=$	$\{\epsilon, [\beta \mapsto G\ \overline{\xi}], \mathbb{D}[\beta] \wedge (G\ \overline{\xi} \sim \beta)\}$
FFLATWL	$canon[w]\ (\mathbb{F}[G\ \overline{\xi}] \sim \tau)$	$=$	$\{\beta, \epsilon, (\mathbb{F}[\beta] \sim \tau) \wedge (G\ \overline{\xi} \sim \beta)\}$
FFLATWR	$canon[w]\ (\tau \sim \mathbb{T}[G\ \overline{\xi}])$		
	where $(\tau = F\ \overline{\xi}'$ or $\tau = tv), \beta$ fresh	$=$	$\{\beta, \epsilon, (\tau \sim \mathbb{T}[\beta]) \wedge (G\ \overline{\xi} \sim \beta)\}$
FFLATGL	$canon[g]\ (\mathbb{F}[G\ \overline{\xi}] \sim \tau)$	$=$	$\{\epsilon, [\beta \mapsto G\ \overline{\xi}], (\mathbb{F}[\beta] \sim \tau) \wedge (G\ \overline{\xi} \sim \beta)\}$
FFLATGR	$canon[g]\ (\tau \sim \mathbb{T}[G\ \overline{\xi}])$		
	where $(\tau = F\ \overline{\xi}'$ or $\tau = tv), \beta$ fresh	$=$	$\{\epsilon, [\beta \mapsto G\ \overline{\xi}], (\tau \sim \mathbb{T}[\beta]) \wedge (G\ \overline{\xi} \sim \beta)\}$

И Т.Д.

Быстрый взгляд на реализацию в компиляторе GHC

Список литературы

Vytiniotis, Dimitrios и др. (сент. 2011). “OutsideIn(x) Modular Type Inference with Local Assumptions”. в: *J. Funct. Program.* 21.4-5. Доступно по адресу https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/jfp-outsidein.pdf, с. 333—412.