

Типы в языках программирования

Лекция 6. Подтипы

Денис Николаевич Москвин

СПбАУ РАН

29.03.2018

- 1 Записи
- 2 Понятие подтипа
- 3 Свойства подтипов
- 4 Подтипы и другие расширения
- 5 Алгоритмическая типизация систем с подтипами

- 1 Записи
- 2 Понятие подтипа
- 3 Свойства подтипов
- 4 Подтипы и другие расширения
- 5 Алгоритмическая типизация систем с подтипами

Новые синтаксические формы

$$\begin{aligned} t ::= & \dots \\ & \{l_i = t_i^{i \in 1 \dots n}\} \\ & t.l \end{aligned}$$
$$\begin{aligned} v ::= & \dots \\ & \{l_i = v_i^{i \in 1 \dots n}\} \end{aligned}$$
$$\begin{aligned} T ::= & \dots \\ & \{l_i : T_i^{i \in 1 \dots n}\} \end{aligned}$$

- Все метки полей должны быть различны.
- Например, двухэлементная запись будет типизироваться так $\{x=0, y=1\} : \{x:\text{Nat}, y:\text{Nat}\}$, а ее первая проекция имеет вид $\{x=0, y=1\}.x$ и типизируется так $\{x=0, y=1\}.x : \text{Nat}$.

Новые правила вычисления

$$\{l_i = v_i^{i \in 1 \dots n}\}.l_j \longrightarrow v_j \quad (\text{E-ProjRcd})$$

$$\frac{t \longrightarrow t'}{t.l \longrightarrow t'.l} \quad (\text{E-Proj})$$

$$\frac{t_j \longrightarrow t'_j}{\{l_i = v_i^{i \in 1 \dots j-1}, l_j = t_j, l_k = t_k^{k \in j+1 \dots n}\} \longrightarrow \{l_i = v_i^{i \in 1 \dots j-1}, l_j = t'_j, l_k = t_k^{k \in j+1 \dots n}\}} \quad (\text{E-Rcd})$$

- Последнее правило обеспечивает приведение полей к значениям слева направо.
- Отметим существование одноэлементной и пустой записи.

Новые правила типизации

$$\frac{\Gamma \vdash t_1 : T_1 \quad \dots \quad \Gamma \vdash t_n : T_n}{\Gamma \vdash \{l_i = t_i^{i \in 1 \dots n}\} : \{l_i : T_i^{i \in 1 \dots n}\}} \quad (T - \text{Rcd})$$

$$\frac{\Gamma \vdash t : \{l_i : T_i^{i \in 1 \dots n}\}}{\Gamma \vdash t.l_j : T_j} \quad (T - \text{Proj})$$

- Первое правило имеет n посылок.
- Записи $\{x=0, y=1\} : \{x : \text{Nat}, y : \text{Nat}\}$ и $\{y=1, x=0\} : \{y : \text{Nat}, x : \text{Nat}\}$ считаются различными и принадлежащими к разным типам. (Возможен более либеральный подход.)
- После введения отношения подтипизации эти записи окажутся эквивалентными.

Записи: сопоставление с образцом

Обобщим сопоставление с образцом с пар на записи.

Модифицированные синтаксические формы

$$\begin{aligned} p &::= x \\ &\quad \{l_i = p_i^{i \in 1 \dots n}\} \\ t &::= \dots \\ &\quad \text{let } p = t \text{ in } t \end{aligned}$$

Модифицированные правила сопоставления

$$\text{match}(x, v) = [x \mapsto v] \quad (\text{M} - \text{Var})$$
$$\frac{\text{match}(p_1, v_1) = \sigma_1 \quad \dots \quad \text{match}(p_n, v_n) = \sigma_n}{\text{match}(\{l_i = p_i^{i \in 1 \dots n}\}, \{l_i = v_i^{i \in 1 \dots n}\}) = \sigma_1 \circ \dots \circ \sigma_n} \quad (\text{M} - \text{Rcd})$$

Последнее правило имеет n посылок.

Правила вычисления

$$\text{let } p = v \text{ in } t \longrightarrow \text{match}(p, v) t \quad (\text{E} - \text{LetV})$$

$$\frac{t_1 \longrightarrow t'_1}{\text{let } p = t_1 \text{ in } t_2 \longrightarrow \text{let } p = t'_1 \text{ in } t_2} \quad (\text{E} - \text{Let})$$

Эти правила сохраняются без изменений.

Записи: сопоставление с образцом: типизация

Отношение типизации для образцов (\Rightarrow) порождает контекст.

Правила типизации для образцов

$$\vdash x : T \Rightarrow x : T \quad (P - \text{Var})$$

$$\frac{\vdash p_1 : T_1 \Rightarrow \Delta_1 \quad \dots \quad \vdash p_n : T_n \Rightarrow \Delta_n}{\vdash \{l_i = p_i^{i \in 1 \dots n}\} : \{l_i : T_i^{i \in 1 \dots n}\} \Rightarrow \Delta_1, \dots, \Delta_n} \quad (P - \text{Rcd})$$

Здесь обобщилось только второе правило.

Правило типизации

$$\frac{\Gamma \vdash t : T \quad \vdash p : T \Rightarrow \Delta \quad \Gamma, \Delta \vdash s : S}{\Gamma \vdash \text{let } p = t \text{ in } s : S} \quad (T - \text{Let})$$

Без изменений, так же как для пар.

Записи: нестрогое сопоставление с образцом

- Можно модифицировать правила сопоставления с образцом для записи таким образом, чтобы
 - образец мог содержать меньше полей, чем передаваемая запись;
 - порядок полей мог бы быть произвольным.
- Тогда, например, для записи $\{lx=1, ly=2, lz=3\}$: $\{lx:Nat, ly:Nat, lz:Nat\}$ можно использовать такое сопоставление $let \{lz=u, lx=v\} = \{lx=1, ly=2, lz=3\} in u^2 - v^2$
- Алгоритмически нужно перебрать все метки в образце, и проверить их наличие в записи.
- Если все проверки удачные — обеспечить связывание, при хотя бы одной неудаче — вернуть неудачу.

Записи: нестрогое правило сопоставления

Было для строгого сопоставления

$$\frac{\forall i \in 1 \dots n. \text{match}(p_i, v_i) = \sigma_i}{\text{match}(\{l_i = p_i^{i \in 1 \dots n}\}, \{l_i = v_i^{i \in 1 \dots n}\}) = \sigma_1 \circ \dots \circ \sigma_n} \quad (\text{M} - \text{Rcd})$$

Стало

Модифицированное правило сопоставления

$$\frac{\begin{array}{c} \{l_i^{i \in 1 \dots n}\} \subseteq \{k_j^{j \in 1 \dots m}\} \\ \forall i \in 1 \dots n. \exists j \in 1 \dots m. l_i = k_j \text{ и } \text{match}(p_i, v_j) = \sigma_i \end{array}}{\text{match}(\{l_i = p_i^{i \in 1 \dots n}\}, \{k_j = v_j^{j \in 1 \dots m}\}) = \sigma_1 \circ \dots \circ \sigma_n} \quad (\text{M} - \text{Rcd}')$$

Здесь $m \geq n$, что обеспечивает возможность делать сопоставление с образцом записи с большим числом полей, чем имеется в образце.

Записи: нестрогая типизация сопоставления

Было для строгого сопоставления

$$\frac{\forall i \in 1 \dots n. \vdash p_i : T_i \Rightarrow \Delta_i}{\vdash \{l_i = p_i^{i \in 1 \dots n}\} : \{l_i : T_i^{i \in 1 \dots n}\} \Rightarrow \Delta_1, \dots, \Delta_n} \quad (P - \text{Rcd})$$

Стало

Модифицированное правило типизации для образцов

$$\frac{\forall i \in 1 \dots n. \exists j \in 1 \dots m. l_i = k_j \text{ и } \vdash p_i : T_j \Rightarrow \Delta_i}{\vdash \{l_i = p_i^{i \in 1 \dots n}\} : \{k_j : T_j^{j \in 1 \dots m}\} \Rightarrow \Delta_1, \dots, \Delta_n} \quad (P - \text{Rcd}')$$

Здесь $m \geq n$, что обеспечивает возможность приписать образцу тип записи с большим, чем необходимо, числом полей.

- 1 Записи
- 2 **Понятие подтипа**
- 3 Свойства подтипов
- 4 Подтипы и другие расширения
- 5 Алгоритмическая типизация систем с подтипами

- Пускай имеется запись $\{x=0, y=1\} : \{x:\text{Nat}, y:\text{Nat}\}$.
- Можем ли мы разумно типизировать следующий терм?

Пример

$(\lambda r : \{x:\text{Nat}\}. r.x) \{x=0, y=1\}$

- Пускай имеется запись $\{x=0, y=1\} : \{x:\text{Nat}, y:\text{Nat}\}$.
- Можем ли мы разумно типизировать следующий терм?

Пример

$(\lambda r : \{x:\text{Nat}\}. r.x) \{x=0, y=1\}$

- Передача аргумента типа $\{x:\text{Nat}, y:\text{Nat}\}$ в функцию, ожидающую тип $\{x:\text{Nat}\}$, всегда безопасна.
- С вычислительной точки зрения проблем нет, но наши правила типизации слишком жесткие, чтобы принять такие конструкции.

- Говорят, что S является *подтипом* T (нотация $S <: T$), подразумевая, что всякий терм типа S можно безопасно использовать в контексте, в котором ожидается тип T .
- Интуитивно: один тип (S) более «информативен», чем другой (T).
- Subset semantics: элементы S являются подмножеством элементов T .

- Отношение $<:$ называют *отношением вложения типов* (subtype relation).
- Правило включения (subsumption):

Правила типизации

$$\frac{\Gamma \vdash t : S \quad S <: T}{\Gamma \vdash t : T} \quad (T - \text{Sub})$$

- Теперь, если отношение вложения допускает $\{x:\text{Nat}, y:\text{Nat}\} <: \{x:\text{Nat}\}$, то $\{x=0, y=1\}:\{x:\text{Nat}\}$ и пример с первого слайда типизируем:

Пример

$(\lambda r:\{x:\text{Nat}\}. r.x) \{x=0, y=1\} : \text{Nat}$

- Зададим правила для отношения вложения $<:$.

Правила вложения

$$S <: S \quad (S - \text{Refl})$$

$$\frac{S <: U \quad U <: T}{S <: T} \quad (S - \text{Trans})$$

$$\{l_i : T_i^{i \in 1 \dots n+k}\} <: \{l_i : T_i^{i \in 1 \dots n}\} \quad (S - \text{RcdWidth})$$

- Последнее правило называется *вложение типов в ширину* (width subtyping): можно «забывать» последние поля записей.
- Более длинный тип соответствует более жесткой, информативной спецификации — ей удовлетворяет меньшее число возможных значений.

Правило вложения (в глубину)

$$\frac{\forall i. S_i <: T_i}{\{l_i : S_i^{i \in 1 \dots n}\} <: \{l_i : T_i^{i \in 1 \dots n}\}} \quad (S - \text{RcdDepth})$$

Упражнения. Покажите, что

- $\{x:\{a:\text{Nat}, b:\text{Nat}\}, y:\{m:\text{Nat}\}\} <: \{x:\{a:\text{Nat}\}, y:\{\}\}$
- $\{x:\{a:\text{Nat}, b:\text{Nat}\}, y:\{m:\text{Nat}\}\} <: \{x:\{a:\text{Nat}\}, y:\{m:\text{Nat}\}\}$
- $\{x:\{a:\text{Nat}, b:\text{Nat}\}, y:\{m:\text{Nat}\}\} <: \{x:\{a:\text{Nat}\}\}$

Правило вложения (перестановка)

$$\frac{\{k_j : S_j^{j \in 1 \dots n}\} \text{ is a permutation of } \{l_i : T_i^{i \in 1 \dots n}\}}{\{k_j : S_j^{j \in 1 \dots n}\} <: \{l_i : T_i^{i \in 1 \dots n}\}} \quad (\text{S} - \text{RcdPerm})$$

Упражнения. Покажите, что

- $\{c:\text{Top}, a:\text{Bool}, b:\text{Nat}\} <: \{a:\text{Bool}, b:\text{Nat}, c:\text{Top}\}$
- $\{a:\text{Bool}, b:\text{Nat}, c:\text{Top}\} <: \{c:\text{Top}, a:\text{Bool}, b:\text{Nat}\}$
- $\{x:\text{Nat}, y:\text{Nat}, z:\text{Nat}\} <: \{y:\text{Nat}\}$

Первые два примера показывают, что антисимметричности нет, то есть нет частичного порядка (только предпорядок).

Правило вложения для функций

- Правило вложения *контравариантно* для типов аргументов, но *ковариантно* для типов результатов.

$$\frac{S_- <: S \quad T <: T_+}{S \rightarrow T <: S_- \rightarrow T_+} \quad (S - \text{Arrow})$$

- Если функция f принимает аргумент типа S , то ей безопасно можно передать любой подтип.

$$\frac{S_- <: S}{S \rightarrow T <: S_- \rightarrow T}$$

- Если функция f возвращает результат типа T , то возвращенное значение безопасно можно использовать вместо любого надтипа.

$$\frac{T <: T_+}{S \rightarrow T <: S \rightarrow T_+}$$

Правило вложения для функций

$$\frac{S_- <: S \quad T <: T_+}{S \rightarrow T <: S_- \rightarrow T_+} \quad (S - \text{Arrow})$$

- Можно использовать $S \rightarrow T$ в любом контексте, где ожидается $S_- \rightarrow T_+$, если аргумент не будет неожиданным функции ($S_- <: S$), а результат не будет неожиданным для контекста вызова ($T <: T_+$).

$\lambda f:T \rightarrow U. \lambda g:S \rightarrow T. \lambda x:S. f (g x)$

$g1 : S_m \rightarrow T_p$

Контравариантность по аргументу: примеры

```
f1 = λr:{x:Nat}. r.x : {x:Nat} → Nat
f2 = λr:{x:Nat,y:Nat}. r.x+r.y : {x:Nat,y:Nat} → Nat

g1 = λf:({x:Nat} → Nat). f {x=1}
g2 = λf:({x:Nat,y:Nat} → Nat). f {x=1,y=2}
```

Какие из следующих вызовов допустимы? $g1\ f1?$ $g2\ f2?$ $g2\ f1?$ $g1\ f2?$

$$\frac{\{x : \text{Nat}, y : \text{Nat}\} <: \{x : \text{Nat}\}}{\{x : \text{Nat}\} \rightarrow \text{Nat} <: \{x : \text{Nat}, y : \text{Nat}\} \rightarrow \text{Nat}} \quad (\text{S} - \text{Arrow})$$

Постройте дерево вывода типа для примера с первого слайда:

```
(λr:{x:Nat}. r.x) {x=0,y=1} : Nat
```


Постройте дерево вывода типа для примера с первого слайда:

$$(\lambda r : \{x : \text{Nat}\}. r.x) \{x=0, y=1\} : \text{Nat}$$

Обозначим $Rx = \{x : \text{Nat}\}$, достройте

$$\frac{\frac{\dots}{(\lambda r : \{x : \text{Nat}\}. r.x) : Rx \rightarrow \text{Nat}} \quad \frac{\dots}{\{x = 0, y = 1\} : Rx}}{(\lambda r : \{x : \text{Nat}\}. r.x) \{x = 0, y = 1\} : \text{Nat}} (\text{T-App})$$

Единственно ли оно?

Постройте дерево вывода типа для примера с первого слайда:

$$(\lambda r : \{x : \text{Nat}\}. r.x) \{x=0, y=1\} : \text{Nat}$$

Обозначим $Rx = \{x : \text{Nat}\}$, достройте

$$\frac{\frac{\dots}{(\lambda r : \{x : \text{Nat}\}. r.x) : Rx \rightarrow \text{Nat}} \quad \frac{\dots}{\{x = 0, y = 1\} : Rx}}{(\lambda r : \{x : \text{Nat}\}. r.x)\{x = 0, y = 1\} : \text{Nat}} (T - \text{App})$$

Единственно ли оно? Нет! Обозначим $Rxy = \{x : \text{Nat}, y : \text{Nat}\}$

$$\frac{\frac{\dots}{(\lambda r : \{x : \text{Nat}\}. r.x) : Rxy \rightarrow \text{Nat}} \quad \frac{\dots}{\{x = 0, y = 1\} : Rxy}}{(\lambda r : \{x : \text{Nat}\}. r.x)\{x = 0, y = 1\} : \text{Nat}} (T - \text{App})$$

Правило вложения для Top

Можно (но не обязательно) ввести наибольший тип.

Правило вложения для Top

$$T <: \text{Top} \quad (S - \text{Top})$$

Упражнения.

- Сколько различных надтипов у типа $\{a:\text{Top}, b:\text{Top}\}$?
- Можно ли найти бесконечную нисходящую цепочку в отношении вложения типов $<:?$
- Можно ли найти бесконечную восходящую цепочку?
- Существует ли тип, являющийся подтипом любого другого типа?
- Существует ли функциональный тип, являющийся надтипом любого другого функционального типа?

- 1 Записи
- 2 Понятие подтипа
- 3 Свойства подтипов**
- 4 Подтипы и другие расширения
- 5 Алгоритмическая типизация систем с подтипами

Лемма (инверсия отношения вложения типов)

- 1 Если $S <: T_1 \rightarrow T_2$, то S имеет вид $S_1 \rightarrow S_2$, причем $T_1 <: S_1$ и $S_2 <: T_2$.
- 2 Если $S <: \{l_i : T_i^{i \in 1 \dots n}\}$, то S имеет вид $\{k_j : S_j^{j \in 1 \dots m}\}$, причем содержит по крайней мере метки полей $\{l_i^{i \in 1 \dots n}\}$, то есть $\{l_i^{i \in 1 \dots n}\} \subseteq \{k_j^{j \in 1 \dots m}\}$, и $S_j <: T_i$ для каждой общей метки $k_j = l_i$.

Лемма (инверсия отношения вложения типов)

- 1 Если $S <: T_1 \rightarrow T_2$, то S имеет вид $S_1 \rightarrow S_2$, причем $T_1 <: S_1$ и $S_2 <: T_2$.
- 2 Если $S <: \{l_i : T_i^{i \in 1 \dots n}\}$, то S имеет вид $\{k_j : S_j^{j \in 1 \dots m}\}$, причем содержит по крайней мере метки полей $\{l_i^{i \in 1 \dots n}\}$, то есть $\{l_i^{i \in 1 \dots n}\} \subseteq \{k_j^{j \in 1 \dots m}\}$, и $S_j <: T_i$ для каждой общей метки $k_j = l_i$.

Лемма (инверсия типизации)

- 1 Если $\Gamma \vdash \lambda x : S_1. s_2 : T_1 \rightarrow T_2$, то $T_1 <: S_1$ и $\Gamma, x : S_1 \vdash s_2 : T_2$.
- 2 Если $\Gamma \vdash \{k_j = s_j^{j \in 1 \dots m}\} : \{l_i : T_i^{i \in 1 \dots n}\}$, то $\{l_i^{i \in 1 \dots n}\} \subseteq \{k_j^{j \in 1 \dots m}\}$ и $\Gamma \vdash s_j : T_i$ для каждой общей метки поля $k_j = l_i$.

Лемма (сохранение типа при подстановке)

Если $\Gamma, x : S \vdash t : T$ и $\Gamma \vdash s : S$, то $\Gamma \vdash [x \mapsto s]t : T$

Доказательство: Индукция по деревьям вывода типов. ■

Теорема о редукции субъекта (сохранении)

Пусть $\Gamma \vdash t : T$ и $t \rightarrow t'$, тогда $\Gamma \vdash t' : T$.

Доказательство: Индукция по дереву вывода. ■

Лемма о канонических формах

1. Если v — замкнутое значение типа $T_1 \rightarrow T_2$, то $v = \lambda x : S_1. t_2$, где $T_1 \leq S_1$.
2. Если v — замкнутое значение типа $\{l_i : T_i^{i \in 1 \dots n}\}$, то v имеет вид $\{k_j = v_j^{j \in 1 \dots m}\}$, причем $\{l_i^{i \in 1 \dots n}\} \subseteq \{k_j^{j \in 1 \dots m}\}$.

Теорема о продвижении

Пусть t — замкнутый, правильно типизированный терм, то есть $\vdash t : T$ для некоторого типа T . Тогда либо t является значением, либо существует некоторый t' , такой, что $t \rightarrow t'$.

Доказательство: Индукция по дереву вывода $t : T$ с использованием леммы о канонических формах.

Добавляется анализ T-Rcd; T-Proj; T-Sub. ■

Можно ввести наименьший тип.

Правило вложения для Bot

$$\text{Bot} <: T \quad (S - \text{Bot})$$

Теорема (Пустота Bot)

Не существует замкнутых значений типа Bot.

Пусть такое v существует.

$$\frac{\Gamma \vdash v : \text{Top} \rightarrow \text{Top} \quad \text{Bot} <: \text{Top} \rightarrow \text{Top}}{\Gamma \vdash v : \text{Bot}} \quad (T - \text{Sub})$$

По лемме о канонических формах v — функция.

$$\frac{\Gamma \vdash v : \{\} \quad \text{Bot} <: \{\}}{\Gamma \vdash v : \text{Bot}} \quad (T - \text{Sub})$$

По лемме о канонических формах v — запись. Противоречие. ■

Правило вложения для Bot

$$\text{Bot} \leq: T \quad (S - \text{Bot})$$

- Плюс — можно повысить до любого типа и типизировать им ошибки.
- Минус — система проверки типов станет сложнее. Правила вроде типизации $(t_1 t_2)$ теперь говорят: t_1 имеет либо стрелочный тип, либо \perp .

- 1 Записи
- 2 Понятие подтипа
- 3 Свойства подтипов
- 4 Подтипы и другие расширения**
- 5 Алгоритмическая типизация систем с подтипами

- У нас имелся оператор приписывания $t \text{ as } T$.
- При наличии подтипов это дает возможность устраивать приведение (casting) типов вверх и вниз.
- При приведении вверх терму приписывается надтип (supertype) того типа, который был бы ему присвоен естественным образом.

$$\frac{\frac{\frac{\vdots}{\Gamma \vdash t : S} \quad \frac{\vdots}{S <: T}}{\Gamma \vdash t : T} \quad (T - \text{Sub})}{\Gamma \vdash t \text{ as } T : T} \quad (T - \text{Ascribe})$$

- Это дает возможность абстракции (T прячет детали S).

Нисходящее приведение

- Нисходящее приведение позволяет приписывать термам типы, которые программа проверки статически вывести не может.

Правило типизации

$$\frac{\Gamma \vdash t : S}{\Gamma \vdash t \text{ as } T : T} \quad (T - \text{Downcast})$$

- Нет никаких требований о связи T и S . Все на совести программиста!
- Контроль происходит во время исполнения.

Правило вычисления

$$\frac{\vdash v : T}{v \text{ as } T \longrightarrow v} \quad (E - \text{Downcast})$$

- Теряем свойство продвижения!

- Для вариантных типов идея обратная: при переходе от подтипа к надтипу метки добавляются, а не отбрасываются.
- Например, $\langle x=0 \rangle : \langle x:\text{Nat}, y:\text{Nat} \rangle$ менее информативен, чем $\langle x=0 \rangle : \langle x:\text{Nat} \rangle$.
- Поэтому $\langle x:\text{Nat} \rangle <: \langle x:\text{Nat}, y:\text{Nat} \rangle$.
- Можем отказаться от нудных точных аннотаций: достаточно писать $\langle l=3 \rangle : \langle l:\text{Nat} \rangle$, точный тип, например $\langle l=3 \rangle : \langle l:\text{Nat}, m:\text{Nat}, b:\text{Bool} \rangle$, будет допустимым надтипом.

- Конструктор List ковариантен:

Правило вложения

$$\frac{S <: T}{\text{List } S <: \text{List } T} \quad (S - \text{List})$$

- Конструктор Ref инвариантен:

Правило вложения

$$\frac{S <: T \quad T <: S}{\text{Ref } S <: \text{Ref } T} \quad (S - \text{Ref})$$

- (Типизированные) каналы для чтения и записи ведут себя с точки зрения типизации аналогично ссылочным ячейкам.
- Они инвариантны; каналы только для чтения — ковариантны, только для записи контравариантны.

- В богатых системах можно ввести отношение вложения для базовых типов.
- Например, $\text{Bool} <: \text{Nat}$, представляя элементы Bool как 0 и 1.
- Тогда вместо $\text{if } b \text{ then } 5 \text{ else } 0$ можно писать $5 * b$.
- Вводя $\text{Int} <: \text{Float}$ в нашей простой «семантике подмножеств», мы требуем буквального вложения целых в плавающие, что не соответствует физическому представлению.
- Решения проблемы: теговая семантика или семантика, основанная на приведении типов (coercion semantics). (см. Пирс 15.6)

- 1 Записи
- 2 Понятие подтипа
- 3 Свойства подтипов
- 4 Подтипы и другие расширения
- 5 Алгоритмическая типизация систем с подтипами

Проблемы декларативной типизации

- Прочитав лемму об инверсии в обратную сторону получить алгоритм вывода не получится.
- Этому мешают следующие правила.

$$\frac{\Gamma \vdash t : S \quad S <: T}{\Gamma \vdash t : T} \quad (T - \text{Sub})$$

$$\frac{S <: U \quad U <: T}{S <: T} \quad (S - \text{Trans})$$

- Проблема с ними в том, что они не содержат указания для какого термина (в первом случае) и типов (во втором) эти правила нужно применять. Во втором случае U нужно еще и угадать!
- Говорят, что эти правила **не** управляются синтаксисом (syntax directed).
- Правило ($S - \text{Ref}$) тоже таково, но слишком простое, чтобы породить серьезные проблемы.

Алгоритмическое отношение вложения типов

$$T <: \text{Top} \quad (\text{SA} - \text{Top})$$

$$\frac{S_- <: S \quad T <: T_+}{S \rightarrow T <: S_- \rightarrow T_+} \quad (\text{SA} - \text{Arrow})$$

$$\frac{\begin{array}{l} \{l_i^{i \in 1 \dots n}\} \subseteq \{k_j^{j \in 1 \dots m}\} \\ \forall i \in 1 \dots n. \exists j \in 1 \dots m. l_i = k_j \text{ и } S_j <: T_i \end{array}}{\{k_j : S_j^{j \in 1 \dots m}\} <: \{l_i : T_i^{i \in 1 \dots n}\}} \quad (\text{SA} - \text{Rcd})$$

- (SA – Rcd) объединяет все три правила для записей.

Лемма

1. Для любого S вложение $S <: S$ выводимо без (S – Refl).
2. Если $S <: T$ выводимо, то оно выводимо без (S – Trans).

- Нам осталось избавиться от правила (T — Sub).
- Эквивалентными преобразованиями дерева вывода можно привести к виду, где (T — Sub) встречается только в двух местах: в конце правого подвывода для термов-аппликаций и в самом конце всего дерева.
- Последнее можно отбросить, это только улучшит результат. Почему?
- Итак, алгоритмическая типизация получится заменой обычного правила для аппликации более мощной версией

Алгоритмическое отношение типизации

$$\frac{\Gamma \Vdash t_1 : T \rightarrow S \quad \Gamma \Vdash t_2 : T_- \quad T_- <: T}{\Gamma \Vdash t_1 t_2 : S} \quad (\text{TA} - \text{App})$$

Теорема (Корректность)

Если $\Gamma \Vdash t : T$, то $\Gamma \vdash t : T$.

Теорема (Полнота, или минимальность типов)

Если $\Gamma \vdash t : T$, то $\Gamma \Vdash t : T_-$, для некоторого $T_- \leq T$.

- Алгоритмические правила присваивают каждому терму его *наименьший* (minimal) тип.
- Например, $\vdash \{x = 0, y = 1\} : \{x : \text{Nat}\}$, но алгоритмически можно получить только $\Vdash \{x = 0, y = 1\} : \{x : \text{Nat}, y : \text{Nat}\}$.

- Правила с ветвями, подобные $(T - If)$, естественно ослабить при наличии подтипов.

$$\frac{\Gamma \vdash t_1 : Bool \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad (T - If)$$

- Есть много способов дать обеим ветвям общий тип.
Например, для

`if true then {x = true, y = false} else {x = false, z = true}`

подойдет и $\{x : Bool\}$, и $\{x : Top\}$ и $\{\}$.

- Раз уж у нас есть предпорядок на типах, разумно выбрать *наименьший* общий надтип.
- Для любой ли пары типов он существует?

Определение

Тип J называется *объединением* (join) пары типов S и T , нотация $S \vee T$, если $S <: J$, $T <: J$, и для любого U , если $S <: U$ и $T <: U$, то $J <: U$.

- Объединение — это точная верхняя грань двух типов, то есть наименьший общий надтип.

Определение

Тип M называется *пересечением* (meet) пары типов S и T , нотация $S \wedge T$, если $M <: S$, $M <: T$, и для любого L , если $L <: S$ и $L <: T$, то $L <: M$.

- Пересечение — это точная нижняя грань двух типов, то есть наибольший общий подтип.

Теорема

1. Для каждой пары типов S и T имеется тип J , такой, что $S \vee T = J$.
2. Для каждой пары типов S и T , имеющих общий подтип, имеется тип M , такой, что $S \wedge T = M$.

- Отношение вложения (в системе с Top и без Bot) обладает объединениями, но не пересечениями.
- Оговорка во второй части теоремы индуцирует понятие *ограниченных пересечений* (bounded meets).
- Доказательство этой теоремы проводится предъявлением взаимно-рекурсивного алгоритма. (См. Пирс, задача 16.3.2.)
- Как вы думаете, откуда берется взаимная рекурсивность?

Поиск объединения и пересечения (задачи)

- Найдите объединения и пересечения для следующих пар типов

$\{1a : \text{Bool}, 1b : \text{Bool}\}$

$\{1b : \text{Bool}, 1c : \text{Bool}\}$

$\{1a : \text{Bool}, 1b : \{\}\}$

$\{1b : \text{Bool}, 1c : \text{Bool}\}$

Bool

$\text{Bool} \rightarrow \text{Bool}$

$\text{Bool} \rightarrow \text{Bool}$

$\text{Bool} \rightarrow \text{Bool} \rightarrow \text{Bool}$

$\{1a : \text{Bool}, 1b : \text{Bool}\} \rightarrow \text{Bool}$

$\{1b : \text{Bool}, 1c : \text{Bool}\} \rightarrow \text{Bool}$

С помощью объединений несложно сконструировать

Алгоритмическое отношение типизации

$$\frac{\Gamma \Vdash t_1 : \text{Bool} \quad \Gamma \Vdash t_2 : T_2 \quad \Gamma \Vdash t_3 : T_3 \quad T_2 \vee T_3 = T}{\Gamma \Vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad (\text{TA} - \text{If})$$

Каков наименьший тип выражения

```
if true then false else {}
```

Хорошо ли это?