Типы в языках программирования
Лекция 6. Подтипы

Денис Николаевич Москвин

ИТМО, корпоративная магистратура JetBrains
Разработка ПО / Software Engineering

24.03.2021

План лекции

План лекции

Записи: синтаксис

Новые синтаксические формы

$$
\begin{array}{rcl}
\mathtt{t} & ::= & \ldots \\
 & & \{\mathtt{l}_i = \mathtt{t}_i^{\,i \in 1 \ldots n}\} \\
 & & \mathtt{t.l} \\
 & & \\
\mathtt{v} & ::= & \ldots \\
 & & \{\mathtt{l}_i = \mathtt{v}_i^{\,i \in 1 \ldots n}\} \\
 & & \\
\mathtt{T} & ::= & \ldots \\
 & & \{\mathtt{l}_i : \mathtt{T}_i^{\,i \in 1 \ldots n}\}
\end{array}
$$

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

Записи: вычисления

Новые правила вычисления

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

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

$$\frac{t_j \longrightarrow t'_j}{\begin{array}{l}\{l_i = v_i^{i \in 1 \ldots j-1}, l_j = t_j, l_k = t_k^{k \in j+1 \ldots n}\} \longrightarrow \\ \{l_i = v_i^{i \in 1 \ldots j-1}, l_j = t'_j, l_k = t_k^{k \in j+1 \ldots n}\}\end{array}} \qquad (E - Rcd)$$

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

Записи: типизация

Новые правила типизации

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

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

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

Записи: сопоставление с образцом

Модифицированные синтаксические формы

$$
\begin{array}{rcl}
\mathtt{p} & ::= & \mathtt{x} \\
 & & \{\mathtt{l}_i = \mathtt{p}_i^{i \in 1 \ldots n}\} \\
\mathtt{t} & ::= & \ldots \\
 & & \mathtt{let\ p = t\ in\ t}
\end{array}
$$

Модифицированные правила сопоставления

$$\mathrm{match}(x, v) = [x \mapsto v] \qquad (\mathrm{M-Var})$$

$$\frac{\mathrm{match}(p_1, v_1) = \sigma_1 \quad \ldots \quad \mathrm{match}(p_n, v_n) = \sigma_n}{\mathrm{match}(\{l_i = p_i^{i \in 1 \ldots n}\}, \{l_i = v_i^{i \in 1 \ldots n}\}) = \sigma_1 \circ \ldots \circ \sigma_n} \qquad (\mathrm{M-Rcd})$$

- Последнее правило имеет n посылок.
- Все переменные в образце должны быть разными.

Записи: сопоставление с образцом: вычисление

Правила вычисления

$$\texttt{let}\ \mathrm{p} = v\ \texttt{in}\ \mathrm{t} \longrightarrow \mathrm{match}(\mathrm{p}, v)\,\mathrm{t} \qquad (\mathrm{E-LetV})$$

$$\frac{\mathrm{t}_1 \longrightarrow \mathrm{t}_1'}{\texttt{let}\ \mathrm{p} = \mathrm{t}_1\ \texttt{in}\ \mathrm{t}_2 \longrightarrow \texttt{let}\ \mathrm{p} = \mathrm{t}_1'\ \texttt{in}\ \mathrm{t}_2} \qquad (\mathrm{E-Let})$$

Эти правила сохраняются без изменений.

Записи: сопоставление с образцом: типизация

Отношение типизации для образцов ($\Rightarrow$) порождает контекст.

Правила типизации для образцов

$$\vdash x : T \;\Rightarrow\; x : T \qquad (P - Var)$$

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

Здесь обобщилось только второе правило.

Правило типизации

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

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

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

Стало

Модифицированное правило сопоставления

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

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

Записи: нестрогая типизация сопоставления

Было для строгого сопоставления

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

Стало

Модифицированное правило типизации для образцов

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

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

План лекции

Избыточная жесткость типа записи

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

Пример

```
(λr:{x:Nat}. r.x) {x=0,y=1}
```

Избыточная жесткость типа записи

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

Пример

```
(λr:{x:Nat}. r.x) {x=0,y=1}
```

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

Принцип безопасной подстановки

- 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 вложения типов

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

Правила типизации

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

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

Пример

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

Определение отношения вложения

- Зададим правила для отношения вложения <:.

Правила вложения

$$\mathtt{S} <: \mathtt{S} \qquad (\mathrm{S-Refl})$$

$$\frac{\mathtt{S} <: \mathtt{U} \qquad \mathtt{U} <: \mathtt{T}}{\mathtt{S} <: \mathtt{T}} \qquad (\mathrm{S-Trans})$$

$$\{\mathtt{l}_i : \mathtt{T}_i^{i \in 1 \ldots n+k}\} <: \{\mathtt{l}_i : \mathtt{T}_i^{i \in 1 \ldots n}\} \qquad (\mathrm{S-RcdWidth})$$

- 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.

Вложение типов в глубину

Правило вложения (в глубину)

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

Упражнения. Покажите, что

- `{x:{a:Nat,b:Nat},y:{m:Nat}} <: {x:{a:Nat},y:{}}`
- `{x:{a:Nat,b:Nat},y:{m:Nat}} <: {x:{a:Nat},y:{m:Nat}}`
- `{x:{a:Nat,b:Nat},y:{m:Nat}} <: {x:{a:Nat}}`

Вложение типов: перестановка

Правило вложения (перестановка)

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

Упражнения. Покажите, что

- `{c:Top,a:Bool,b:Nat} <: {a:Bool,b:Nat,c:Top}`
- `{a:Bool,b:Nat,c:Top} <: {c:Top,a:Bool,b:Nat}`
- `{x:Nat,y:Nat,z:Nat} <: {y: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.

$$\frac{\mathtt{S}_- <: \mathtt{S} \qquad \mathtt{T} <: \mathtt{T}_+}{\mathtt{S} \to \mathtt{T} <: \mathtt{S}_- \to \mathtt{T}_+} \quad (\mathrm{S-Arrow})$$

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

$$\frac{\mathtt{S}_- <: \mathtt{S}}{\mathtt{S} \to \mathtt{T} <: \mathtt{S}_- \to \mathtt{T}}$$

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

$$\frac{\mathtt{T} <: \mathtt{T}_+}{\mathtt{S} \to \mathtt{T} <: \mathtt{S} \to \mathtt{T}_+}$$

Правило вложения для функций (2)

Правило вложения для функций

$$\frac{\mathtt{S}_- <: \mathtt{S} \qquad \mathtt{T} <: \mathtt{T}_+}{\mathtt{S} \to \mathtt{T} <: \mathtt{S}_- \to \mathtt{T}_+} \quad (\mathrm{S-Arrow})$$

- Можно использовать $\mathtt{S} \to \mathtt{T}$ в любом контексте, где ожидается $\mathtt{S}_- \to \mathtt{T}_+$, если аргумент не будет неожиданным функции ($\mathtt{S}_- <: \mathtt{S}$), а результат не будет неожиданным для контекста вызова ($\mathtt{T} <: \mathtt{T}_+$).

```
λf:Tp→U. λg:Sm→Tp. λx:Sm. f (g x)

g0 : S→T
```

- Безопасно использовать g0 вместо g.
- По правилу вложения можем приписать g0 : Sm→Tp.

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`?

$$\frac{\{x : \mathtt{Nat}, y : \mathtt{Nat}\} <: \{x : \mathtt{Nat}\}}{\{x : \mathtt{Nat}\} \to \mathtt{Nat} <: \{x : \mathtt{Nat}, y : \mathtt{Nat}\} \to \mathtt{Nat}} \quad (\mathrm{S-Arrow})$$

Задача

Постройте дерево вывода типа для примера с первого слайда:

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

Задача

Постройте дерево вывода типа для примера с первого слайда:

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

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

$$\frac{\dfrac{\cdots}{(\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\ \mathtt{r.x}) : \mathtt{Rx} \to \mathtt{Nat}} \qquad \dfrac{\cdots}{\{\mathtt{x} = 0, \mathtt{y} = 1\} : \mathtt{Rx}}}{(\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\ \mathtt{r.x})\{\mathtt{x} = 0, \mathtt{y} = 1\} : \mathtt{Nat}}(\mathrm{T} - \mathrm{App})$$

Единственно ли оно?

Задача

Постройте дерево вывода типа для примера с первого слайда:

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

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

$$\frac{\dfrac{\dots}{(\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\ \mathtt{r.x}) : \mathtt{Rx} \to \mathtt{Nat}} \qquad \dfrac{\dots}{\{\mathtt{x} = 0, \mathtt{y} = 1\} : \mathtt{Rx}}}{(\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\ \mathtt{r.x})\{\mathtt{x} = 0, \mathtt{y} = 1\} : \mathtt{Nat}}(\mathrm{T-App})$$

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

$$\frac{\dfrac{\dots}{(\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\ \mathtt{r.x}) : \mathtt{Rxy} \to \mathtt{Nat}} \qquad \dfrac{\dots}{\{\mathtt{x} = 0, \mathtt{y} = 1\} : \mathtt{Rxy}}}{(\lambda \mathtt{r} : \{\mathtt{x} : \mathtt{Nat}\}.\ \mathtt{r.x})\{\mathtt{x} = 0, \mathtt{y} = 1\} : \mathtt{Nat}}(\mathrm{T-App})$$

Правило вложения для `Top`

Можно (но не обязательно) ввести наибольший тип.

Правило вложения для `Top`

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

Упражнения.

- Сколько различных надтипов у типа `{a:Top,b: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?

План лекции

Леммы об инверсии

Лемма (инверсия отношения вложения типов)

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

Леммы об инверсии

Лемма (инверсия отношения вложения типов)

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

Лемма (инверсия типизации)

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

Сохранение

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 $t \to t'$, тогда $\Gamma \vdash t' : T$.

Доказательство: Индукция по дереву вывода. ■

Продвижение

Лемма о канонических формах

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

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 $t \to t'$.

Доказательство: Индукция по дереву вывода $t : T$ с использованием леммы о канонических формах.
Добавляется анализ T-Rcd; T-Proj; T-Sub.■

Тип Bot

Можно ввести наименьший тип.

Правило вложения для Bot

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

Теорема (Пустота Bot)

Не существует замкнутых значений типа Bot.

Пусть такое v существует.

$$\frac{\Gamma \vdash v : \texttt{Bot} \qquad \texttt{Bot} <: \texttt{Top} \to \texttt{Top}}{\Gamma \vdash v : \texttt{Top} \to \texttt{Top}} \quad (\mathrm{T-Sub})$$

По лемме о канонических формах v — функция.

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

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

Прагматика Bot

Правило вложения для Bot

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

- Плюс — можно повысить до любого типа и типизировать им ошибки.
- Минус — система проверки типов станет сложнее. Правила вроде типизации $(\mathrm{t}_1\,\mathrm{t}_2)$ теперь говорят: t_1 имеет либо стрелочный тип, либо $\bot$.

План лекции

Подтипы и приписывание

- У нас имелся оператор приписывания `t 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.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathtt{t} : \mathtt{S}} \qquad \dfrac{\vdots}{\mathtt{S} <: \mathtt{T}}}{\Gamma \vdash \mathtt{t} : \mathtt{T}}\ (\mathrm{T-Sub})$$

$$\dfrac{\Gamma \vdash \mathtt{t} : \mathtt{T}}{\Gamma \vdash \mathtt{t\ as\ T} : \mathtt{T}}\ (\mathrm{T-Ascribe})$$

- 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

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

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

Правило вычисления

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

- Теряем свойство продвижения!

Вариантные типы

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

Списки, ссылки, каналы

- Конструктор `List` ковариантен:

Правило вложения

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

- Конструктор `Ref` инвариантен:

Правило вложения

$$\frac{\mathtt{S} <: \mathtt{T} \qquad \mathtt{T} <: \mathtt{S}}{\mathtt{Ref\ S} <: \mathtt{Ref\ T}} \quad (\mathrm{S-Ref})$$

- (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.

Базовые типы

- В богатых системах можно ввести отношение вложения для базовых типов.
- Например, `Bool <: Nat`, представляя элементы `Bool` как 0 и 1.
- Тогда вместо `if b then 5 else 0` можно писать `5 * b`.
- Вводя `Int <: 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)

План лекции

Проблемы декларативной типизации

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

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

$$\frac{\mathtt{S} <: \mathtt{U} \qquad \mathtt{U} <: \mathtt{T}}{\mathtt{S} <: \mathtt{T}} \quad (\mathrm{S-Trans})$$

- Проблема с ними в том, что они не содержат указания для какого терма (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 $(\mathrm{S-Refl})$ тоже таково, но слишком простое, чтобы порождать серьезные проблемы.

Алгоритмическое отношение вложения типов

Алгоритмическое отношение вложения типов

$$\mathtt{T} <: \mathtt{Top} \qquad (\mathrm{SA-Top})$$

$$\frac{\mathtt{S}_- <: \mathtt{S} \qquad \mathtt{T} <: \mathtt{T}_+}{\mathtt{S} \to \mathtt{T} <: \mathtt{S}_- \to \mathtt{T}_+} \qquad (\mathrm{SA-Arrow})$$

$$\frac{\{\mathtt{l}_i^{i\in 1\ldots n}\} \subseteq \{\mathtt{k}_j^{j\in 1\ldots m}\} \quad \forall^{i\in 1\ldots n}.\ \exists^{j\in 1\ldots m}.\ \mathtt{l}_i = \mathtt{k}_j \text{ и } \mathtt{S}_j <: \mathtt{T}_i}{\{\mathtt{k}_j : \mathtt{S}_j^{j\in 1\ldots m}\} <: \{\mathtt{l}_i : \mathtt{T}_i^{i\in 1\ldots n}\}} \qquad (\mathrm{SA-Rcd})$$

- $(\mathrm{SA-Rcd})$ объединяет все три правила для записей (вложение в ширину, глубину и перестановка).
- Эквивалентность с декларативным отношением вложения доказывается индукцией по деревьям вывода.

Алгоритмическое отношение вложения типов (2)

Правила рефлексивности и транзитивности в новой системе несущественны.

Лемма

Для любого $\mathtt{S}$ вложение $\mathtt{S} <: \mathtt{S}$ выводимо без $(\mathrm{S} - \mathrm{Refl})$.

Доказательство: индукция по структуре $\mathtt{S}$. ■

- При наличии `Bool` нужно будет добавить $\mathtt{Bool} <: \mathtt{Bool}$.
- И так для всех базовых типов.

Лемма

Если $\mathtt{S} <: \mathtt{T}$ выводимо, то оно выводимо без $(\mathrm{S} - \mathrm{Trans})$.

Доказательство: индукция по размеру дерева вывода $\mathtt{S} <: \mathtt{T}$. ■

Алгоритмическое отношение типизации

- Нам осталось избавится от правила $(\mathrm{T} - \mathrm{Sub})$.
- Эквивалентными преобразованиями деревья вывода можно привести к виду, где $(\mathrm{T} - \mathrm{Sub})$ 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

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

Связь алгоритмической и декларативной типизации

Теорема (Корректность)

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

Теорема (Полнота, или минимальность типов)

Если $\Gamma \vdash \mathtt{t} : \mathtt{T}$, то $\Gamma \Vdash \mathtt{t} : \mathtt{T}_{\_}$, для некоторого $\mathtt{T}_{\_} <: \mathtt{T}$.

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

Проблема ветвления результатов

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \quad (\mathrm{T}-\mathrm{If})$$

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

  $\mathtt{if\ true\ then}\ \{\mathtt{x} = \mathtt{true}, \mathtt{y} = \mathtt{false}\}\ \mathtt{else}\ \{\mathtt{x} = \mathtt{false}, \mathtt{z} = \mathtt{true}\}$

  подойдет и $\{\mathtt{x} : \mathtt{Bool}\}$, и $\{\mathtt{x} : \mathtt{Top}\}$ и $\{\}$.
- Раз уж у нас есть предпорядок на типах, разумно выбрать *наименьший* общий надтип.
- Для любой ли пары типов он существует?

Типы-объединения и типы-пересечения

Определение

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

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

Определение

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

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

Поиск объединения

Теорема

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

- 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

$\{\texttt{la} : \texttt{Bool}, \texttt{lb} : \texttt{Bool}\}$	$\{\texttt{lb} : \texttt{Bool}, \texttt{lc} : \texttt{Bool}\}$
$\{\texttt{la} : \texttt{Bool}, \texttt{lb} : \{\}\}$	$\{\texttt{lb} : \texttt{Bool}, \texttt{lc} : \texttt{Bool}\}$
$\texttt{Bool}$	$\texttt{Bool} \to \texttt{Bool}$
$\texttt{Bool} \to \texttt{Bool}$	$\texttt{Bool} \to \texttt{Bool} \to \texttt{Bool}$
$\{\texttt{la} : \texttt{Bool}, \texttt{lb} : \texttt{Bool}\} \to \texttt{Bool}$	$\{\texttt{lb} : \texttt{Bool}, \texttt{lc} : \texttt{Bool}\} \to \texttt{Bool}$

Алгоритмическая типизация ветвления

С помощью объединений несложно сконструировать

Алгоритмическое отношение типизации

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

Каков наименьший тип выражения

$$\mathtt{if\ true\ then\ false\ else}\ \{\}$$

Хорошо ли это?