Типы в языках программирования
Лекция 1. Язык арифметических выражений

Денис Николаевич Москвин

СПбАУ РАН

15.02.2018

План лекции

1. Что такое типы
2. Бестиповая арифметика
3. Арифметика с типами

План лекции

1. Что такое типы
2. Бестиповая арифметика
3. Арифметика с типами

Системы типов

Два подхода к типам:

- Теория типов как раздел логики. (Рассел - Рамсей/Черч - Карри/Говард - Мартин-Лёф - Берарди/Терлоу/Барендрегт - Воеводский)
- Типы в прикладной информатике. (Системы типов для языков программирования.)

Система типов — это гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.

Бенджамин Пирс

Для чего нужны типы?

- Выявление некоторых классов ошибок.

```
GHCi> length 42
GHCi> foldr (+) "ABCDE"
```

- Обеспечивают механизм *абстракции*, позволяя отделить протокол использования от деталей реализации.
- Обеспечивают *безопасность языка*, через механизм целостности абстракций.
- Документация.
- Эффективность.

Какие бывают системы типов?

Возможны классификации систем типам по разным аспектам:

- статические (static) vs динамические (dinamic);
- явные (explicit) vs неявные (implicit);
- сильные (strong) vs слабые (weak);
- структурные (structural) vs именные (nominal).
- консервативные (conservative) vs выразительные (expressive).

Пример слабой системы:

```
x = 5;
y = "37";
z = x + y;
```

Когда изготавливать систему типов?

Четыре взгляда на языки программирования:

- точка зрения прикладного программиста;
- точка зрения разработчика библиотек, компиляторов, инфраструктуры;
- точка зрения разработчиков стандарта существующего языка;
- точка зрения разработчиков нового языка.

Попытки приделать статическую систему типов на уровнях, отличных от последнего, обычно не очень удачны.

План лекции

Термы исчисления

В стандартной форме Бэкуса-Наура:

Термы (выражения) исчисления

```
t ::=
      true
      false
      if t then t else t
      0
      succ t
      pred t
      iszero t
```

Здесь `t` — метапеременная.

Можно описать и *индуктивно*, и в виде *схем правил вывода*, и в виде *конкретной иерархии*.

Термы исчисления, индуктивно

Термы, индуктивно

Множество термов T — это наименьшее множество, обладающее следующими свойствами:

$$\{\mathtt{true}, \mathtt{false}, 0\} \subset T$$

$$\mathtt{t} \in T \;\Rightarrow\; \{\mathtt{succ}\ \mathtt{t}, \mathtt{pred}\ \mathtt{t}, \mathtt{iszero}\ \mathtt{t}\} \subset T$$

$$\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3 \in T \;\Rightarrow\; \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \in T$$

Почему мы не говорим о скобках, хотя, вроде, надо бы?

Термы исчисления, индуктивно

Термы, индуктивно

Множество термов T — это наименьшее множество, обладающее следующими свойствами:

$$\{\mathtt{true}, \mathtt{false}, 0\} \subset T$$

$$\mathtt{t} \in T \;\Rightarrow\; \{\mathtt{succ}\ \mathtt{t}, \mathtt{pred}\ \mathtt{t}, \mathtt{iszero}\ \mathtt{t}\} \subset T$$

$$\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3 \in T \;\Rightarrow\; \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \in T$$

Почему мы не говорим о скобках, хотя, вроде, надо бы?

Мы определяем термы как деревья в абстрактном синтаксисе. При записи в линейном, строковом виде используем группирующие скобки.

Термы исчисления, конкретно

Конкретная иерархия

$$
\begin{array}{lcll}
T_0 & = & \varnothing & \\
T_{i+1} & = & & \{\texttt{true}, \texttt{false}, 0\} \\
 & & \cup & \{\texttt{succ t}, \texttt{pred t}, \texttt{iszero t} \mid \texttt{t} \in T_i\} \\
 & & \cup & \{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 \mid \texttt{t}_1, \texttt{t}_2, \texttt{t}_3 \in T_i\} \\
 & & & \\
T & = & \bigcup_i T_i &
\end{array}
$$

Сколько элементов содержит T_1? T_2?

Является ли эта иерархия куммулятивной?

Индуктивное определение множества констант терма:

Определение

$$
\begin{array}{lcl}
cnst(\mathtt{true}) & = & \{\mathtt{true}\} \\
cnst(\mathtt{false}) & = & \{\mathtt{false}\} \\
cnst(\mathtt{0}) & = & \{\mathtt{0}\} \\
cnst(\mathtt{succ\ t}) & = & cnst(\mathtt{t}) \\
cnst(\mathtt{pred\ t}) & = & cnst(\mathtt{t}) \\
cnst(\mathtt{iszero\ t}) & = & cnst(\mathtt{t}) \\
cnst(\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3) & = & cnst(\mathtt{t}_1) \cup cnst(\mathtt{t}_2) \cup cnst(\mathtt{t}_3)
\end{array}
$$

Определите *size* (количество узлов всех типов) и *depth* (глубина дерева).

Принцип структурной индукции

Если из того, что свойство выполнено для всех непосредственных подтермов данного терма выводимо, что свойство выполнено для данного терма, то свойство верно для любого терма.

Лемма

$$\forall \mathrm{t}\ |\mathit{cnst}(\mathrm{t})| \leqslant \mathit{size}(\mathrm{t})$$

Доказываем структурной индукцией, перебирая всевозможные синтаксические формы.
Можно доказывать и индукцией по глубине и по размеру.

Семантические стили

- **Операционная семантика:** описываем *абстрактную машину*. Для простых языков *состояние* это терм, а поведение задается *функцией перехода*, которая либо описывает следующее состояние, либо говорит, что достигнуто конечное состояние. *Смысл* терма — его конечное состояние.
- **Денотационная семантика:** смысл терма — некоторый математический объект из *семантического домена*. Для каждого терма задается *функция интерпретации*.

Для того, чтобы описать процесс вычисления, надо

- задать подмножество термов, называемых *значениями*;
- задать отношение *вычисления за один шаг*: $t \longrightarrow t'$.

На булевом подмножестве

Термы и значения

```
t ::=
      true
      false
      if t then t else t
v ::=
      true
      false
```

Вычисление

$$\texttt{if true then } \mathtt{t}_2 \texttt{ else } \mathtt{t}_3 \;\rightarrow\; \mathtt{t}_2 \qquad (\mathrm{E-IfTrue})$$

$$\texttt{if false then } \mathtt{t}_2 \texttt{ else } \mathtt{t}_3 \;\rightarrow\; \mathtt{t}_3 \qquad (\mathrm{E-IfFalse})$$

$$\frac{\mathtt{t}_1 \rightarrow \mathtt{t}_1'}{\texttt{if } \mathtt{t}_1 \texttt{ then } \mathtt{t}_2 \texttt{ else } \mathtt{t}_3 \;\rightarrow\; \texttt{if } \mathtt{t}_1' \texttt{ then } \mathtt{t}_2 \texttt{ else } \mathtt{t}_3} \qquad (\mathrm{E-If})$$

Как будет вычисляться следующее выражение?

```
if true then (if false then false else false) else true
```

Можно построить дерево вывода для вычислений: листья — экземпляры правил $(\mathrm{E-IfTrue})$ и $(\mathrm{E-IfFalse})$, узлы — $(\mathrm{E-If})$. (При этом ветвлений нет!)

Детерминированность одношагового вычисления

Теорема

Если $\mathtt{t} \to \mathtt{t}'$ и $\mathtt{t} \to \mathtt{t}''$, то $\mathtt{t}' = \mathtt{t}''$.

Доказательство: индукция по дереву вывода для вычисления $\mathtt{t} \to \mathtt{t}'$. Смотрим в корень, разбираем возможные варианты структуры $\mathtt{t}$, исходя из последнего правила, затем сравниваем с деревом для $\mathtt{t} \to \mathtt{t}''$.

- $(\mathrm{E-IfTrue})$: $\mathtt{t} = \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$ и $\mathtt{t}_1 = \mathtt{true}$. В дереве вывода для $\mathtt{t} \to \mathtt{t}''$ не может быть другого правила.
- $(\mathrm{E-IfFalse})$: аналогично.
- $(\mathrm{E-If})$: аналогично, с использованием IH.

■

Нормальная форма

Определение

Если к терму неприменимо ни одно вычислительное правило, то говорят, что он находится в *нормальной форме*.

Теорема

Любое значение является нормальной формой.

Теорема

Если $\mathtt{t}$ — нормальная форма, то $\mathtt{t}$ является значением.

Доказательство: пусть $\mathtt{t}$ не значение, докажем, что не NF, структурной индукцией. «Не значение» должно иметь вид $\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3$. $\mathtt{t}_1 = \mathtt{false}$ или $\mathtt{t}_1 = \mathtt{true}$ — не NF, в третьем случае пользуемся IH. ■

Второе утверждение верно только для нашего простого исчисления, первое должно выполнятся для любого разумного.

Многошаговое вычисление

Определение

Отношение многошагового вычисления $t \twoheadrightarrow t'$ — это рефлексивно-транзитивное замыкание отношения (одношагового) вычисления.

Теорема (единственность нормальной формы)

Если u и v — нормальные формы, и $t \twoheadrightarrow u$ и $t \twoheadrightarrow v$, то $u = v$.

Доказательство: следует из детерминированности одношагового вычисления. ■

Завершимость вычисления

Теорема

Для каждого терма t существует нормальная форма t' такая, что $t \twoheadrightarrow t'$.

Доказательство: На каждом шаге вычисления размер терма сокращается. ■

(Эта теорема верна только для ограниченного круга исчислений.)

Расширение до арифметики

Термы и значения

```
t ::= ...
       0
       succ t
       pred t
       iszero t
v ::= ...
       nv
nv ::=
       0
       succ nv
```

Введена синтаксическая категория числовых значений `nv`.

Вычисление (новые правила)

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{succ}\ \mathtt{t}_1 \ \to\ \mathtt{succ}\ \mathtt{t}_1'} \qquad (\mathrm{E-Succ})$$

$$\mathtt{pred}\ 0 \ \to\ 0 \qquad (\mathrm{E-PredZero})$$

$$\mathtt{pred}(\mathtt{succ}\ nv_1) \ \to\ nv_1 \qquad (\mathrm{E-PredSucc})$$

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{pred}\ \mathtt{t}_1 \ \to\ \mathtt{pred}\ \mathtt{t}_1'} \qquad (\mathrm{E-Pred})$$

$$\mathtt{iszero}\ 0 \ \to\ \mathtt{true} \qquad (\mathrm{E-IsZeroZero})$$

$$\mathtt{iszero}(\mathtt{succ}\ nv_1) \ \to\ \mathtt{false} \qquad (\mathrm{E-IsZeroSucc})$$

$$\frac{\mathtt{t}_1 \to \mathtt{t}_1'}{\mathtt{iszero}\ \mathtt{t}_1 \ \to\ \mathtt{iszero}\ \mathtt{t}_1'} \qquad (\mathrm{E-IsZero})$$

Детерминировано ли вычисление `pred(succ(pred 0))`?

Детерминированность одношагового вычисления

Для нашей расширенной системы по-прежнему верна теорема о детерминированности вычислений.

Теорема

Если $\mathrm{t} \to \mathrm{t}'$ и $\mathrm{t} \to \mathrm{t}''$, то $\mathrm{t}' = \mathrm{t}''$.

- Можно расширить вычисления для многошаговых и ввести понятие нормальной формы.
- Верна ли теорема о завершимости?

Детерминированность одношагового вычисления

Для нашей расширенной системы по-прежнему верна теорема о детерминированности вычислений.

Теорема

Если $t \to t'$ и $t \to t''$, то $t' = t''$.

- Можно расширить вычисления для многошаговых и ввести понятие нормальной формы.
- Верна ли теорема о завершимости? Да.

Детерминированность одношагового вычисления

Для нашей расширенной системы по-прежнему верна теорема о детерминированности вычислений.

Теорема

Если $\mathrm{t} \to \mathrm{t}'$ и $\mathrm{t} \to \mathrm{t}''$, то $\mathrm{t}' = \mathrm{t}''$.

- Можно расширить вычисления для многошаговых и ввести понятие нормальной формы.
- Верна ли теорема о завершимости? Да.
- Верна ли теорема о том, всякая нормальная форма является значением?

Детерминированность одношагового вычисления

Для нашей расширенной системы по-прежнему верна теорема о детерминированности вычислений.

Теорема

Если $\mathrm{t} \to \mathrm{t}'$ и $\mathrm{t} \to \mathrm{t}''$, то $\mathrm{t}' = \mathrm{t}''$.

- Можно расширить вычисления для многошаговых и ввести понятие нормальной формы.
- Верна ли теорема о завершимости? Да.
- Верна ли теорема о том, всякая нормальная форма является значением? Нет!

Тупиковое состояние

Определение

Терм называется *тупиковым* (stuck), если он находится в нормальной форме, но не является значением.

Пример тупикового терма

```
succ true
```

Для нашей абстрактной машины тупиковое состояние это ошибка времени исполнения.

Семантика с большим шагом

Семантика с большим шагом описывает вычислительные правила в посылках (и заключениях) через понятие «терм $\mathtt{t}$ имеет при вычислении значение v», нотация $\mathtt{t} \Downarrow v$.

Вычисление

$$v \Downarrow v \qquad (\mathrm{B-Value})$$

$$\frac{\mathtt{t}_1 \Downarrow \mathtt{true} \qquad \mathtt{t}_2 \Downarrow v_2}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \ \Downarrow \ v_2} \qquad (\mathrm{B-IfTrue})$$

$$\frac{\mathtt{t}_1 \Downarrow \mathtt{false} \qquad \mathtt{t}_3 \Downarrow v_3}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \ \Downarrow \ v_3} \qquad (\mathrm{B-IfFalse})$$

$$\ldots$$

Упражнение: продолжите самостоятельно.

Упражнение

Предположим, что нам захотелось поменять стратегию вычисления для булева языка так, чтобы ветви `then` и `else` в условном выражении вычислялись (в указанном порядке) до того, как вычислится само условие.
Покажите, как нужно изменить правила вычисления, чтобы добиться такого поведения.

План лекции

Бестиповая арифметика

Термы и значения

```
t ::=
      true
      false
      if t then t else t
      0
      succ t
      pred t
      iszero t
v ::=
      true
      false
      nv
nv ::=
      0
      succ nv
```

Типы

- У нас имелись тупиковые термы, вроде `pred false`.
- Хотелось бы иметь возможность *статически* проверять, зайдет ли вычисление в тупик.
- Типы позволят это сделать, но *консервативно*, то есть отбросив при этом и некоторые нетупиковые, например `if true then 0 else false`.

Введем новые синтаксические формы

Типы

```
T ::=
    Bool
    Nat
```

Отношение типизации

Правила типизации

$$\texttt{true} : \texttt{Bool} \qquad (T - \mathsf{True})$$

$$\texttt{false} : \texttt{Bool} \qquad (T - \mathsf{False})$$

$$\frac{\texttt{t}_1 : \texttt{Bool} \quad \texttt{t}_2 : \texttt{T} \quad \texttt{t}_3 : \texttt{T}}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}} \qquad (T - \mathsf{If})$$

$$0 : \texttt{Nat} \qquad (T - \mathsf{Zero})$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{succ}(\texttt{t}_1) : \texttt{Nat}} \qquad (T - \mathsf{Succ})$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{pred}(\texttt{t}_1) : \texttt{Nat}} \qquad (T - \mathsf{Pred})$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{iszero}(\texttt{t}_1) : \texttt{Bool}} \qquad (T - \mathsf{IsZero})$$

Отношение типизации

Определение

Отношение типизации (typing relation) для арифметических выражений — это наименьшее бинарное отношение между термами и типами, удовлетворяющее всем правилам с предыдущего слайда.

Определение

Терм $\mathtt{t}$ называется ***типизируемым*** (typable) (или ***корректно типизированным***, well-typed), если существует тип $\mathtt{T}$ такой, что $\mathtt{t} : \mathtt{T}$.

Лемма об инверсии

Лемма (генерации)

1. Если $\mathtt{true} : R$, то $R = \mathtt{Bool}$.
2. Если $\mathtt{false} : R$, то $R = \mathtt{Bool}$.
3. Если $\mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : R$, то $t_1 : \mathtt{Bool}$, $t_2 : R$ и $t_3 : R$.
4. Если $0 : R$, то $R = \mathtt{Nat}$.
5. Если $\mathtt{succ}(t_1) : R$, то $R = \mathtt{Nat}$ и $t_1 : \mathtt{Nat}$.
6. Если $\mathtt{pred}(t_1) : R$, то $R = \mathtt{Nat}$ и $t_1 : \mathtt{Nat}$.
7. Если $\mathtt{iszero}(t_1) : R$, то $R = \mathtt{Bool}$ и $t_1 : \mathtt{Nat}$.

Лемма об инверсии дает нам рекурсивный алгоритм вывода типов.

Теорема

Все подтермы типизируемого терма типизируемы.

Дерево вывода типа

Правила типизации позволяют строить дерево вывода типа.

Докажем, например, что

`if iszero 0 then 0 else pred 0 : Nat`.

Единственность типа

Теорема о единственности типа

Всякий терм t имеет не более одного типа.

Доказательство: Структурная индукция по t, с использованием леммы генерации. ■

Это свойство выполняется далеко не для всех систем.

Корректность типа (типобезопасность)

Безопасность = продвижение + сохранение (Харпер)

- Продвижение: Правильно типизированный терм не может быть тупиковым (либо это значение, либо может быть проделан следующий шаг в соответствии с правилами вычисления).
- Сохранение: Если над правильно типизированным термом выполнить шаг вычисления, то получившийся терм также правильно типизирован.

Второе свойство почти всегда можно сформулировать более сильным образом — тип сохраняется при вычислениях.

Канонические формы

Определение

Канонические формы (canonical forms) для некоторого типа — это правильно типизированные значения этого типа.

Лемма о канонических формах

1. Если v — значение типа `Bool`, то v равно либо `true`, либо `false`.
2. Если v — значение типа `Nat`, то v является числовым значением.

Доказательство:
1. Значения — это `true`, `false`, `0` или `succ(nv)`. Первые два дают искомые КФ, а вторые два имеют тип `Nat`.
2. Самостоятельно. ■

Продвижение

Теорема о продвижении

Пусть $\mathtt{t} : \mathtt{T}$. Тогда либо $\mathtt{t}$ является значением, либо существует некоторый $\mathtt{t}'$, такой, что $\mathtt{t} \longrightarrow \mathtt{t}'$.

Доказательство: Индукция по дереву вывода $\mathtt{t} : \mathtt{T}$ с использованием леммы о канонических формах. ■

Сохранение

Теорема о редукции субъекта (сохранении)

Пусть $\mathtt{t} : \mathtt{T}$ и $\mathtt{t} \to \mathtt{t}'$, тогда $\mathtt{t}' : \mathtt{T}$.

Доказательство: Индукция по дереву вывода $\mathtt{t} : \mathtt{T}$ с анализом соответствующих вычислительных правил. ■