Types in Programming Languages (10)

Christian Urban

http://www4.in.tum.de/lehre/vorlesungen/types/WS0607/

Recap from last Week

- We had a look at Featherweight Java (its type-system and transition relation). I assume you did your homework and re-read the chapter by Pierce.
- We briefly talked about the Curry-Howard correspondence. We will have a closer look at this today.

Lambda-Calculus

- Extremely small Turing-complete programming language.
- Church-numerals are an encoding of numbers to lambda-terms:

$$\begin{array}{lcl} \mathbf{0} & \mapsto & \boldsymbol{\lambda} f\, x.\, x \\ \mathbf{1} & \mapsto & \boldsymbol{\lambda} f\, x.\, f x \\ \mathbf{2} & \mapsto & \boldsymbol{\lambda} f\, x.\, f(f x) \\ \mathbf{3} & \mapsto & \boldsymbol{\lambda} f\, x.\, f(f(f x)) \\ & \vdots & \end{array}$$

- Addition: $\boldsymbol{\lambda} m\, n\, f\, x.\, m f(n f x)$

3+2

$$(\lambda m\, n\, f\, x.\; m f (n f x))\ (\lambda f\, x.\; f^3 x)\ (\lambda f\, x.\; f^2 x)$$

3+2

$$(\lambda m\, n\, f\, x.\ m f (n f x))\ (\lambda f\, x.\ f^3 x)\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda n\, f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ (n f x))\ (\lambda f\, x.\ f^2 x)$$

3+2

$$(\lambda m\, n\, f\, x.\ m f (n f x))\ (\lambda f\, x.\ f^3 x)\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda n\, f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ (n f x))\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ ((\lambda f\, x.\ f^2 x)\ f\ x))$$

3+2

$$(\lambda m\, n\, f\, x.\ m f (n f x))\ (\lambda f\, x.\ f^3 x)\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda n\, f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ (n f x))\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ ((\lambda f\, x.\ f^2 x)\ f\ x))$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ ((\lambda x.\ f^2 x)\ x))$$

3+2

$$(\lambda m\, n\, f\, x.\ m f (n f x))\ (\lambda f\, x.\ f^3 x)\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda n\, f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ (n f x))\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ ((\lambda f\, x.\ f^2 x)\ f\ x))$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ ((\lambda x.\ f^2 x)\ x))$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ (f^2 x))$$

3+2

$$(\lambda m\, n\, f\, x.\ mf(nfx))\ (\lambda f\, x.\ f^3x)\ (\lambda f\, x.\ f^2x)$$

$$(\lambda n\, f\, x.\ (\lambda f\, x.\ f^3x)\ f\ (nfx))\ (\lambda f\, x.\ f^2x)$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3x)\ f\ ((\lambda f\, x.\ f^2x)\ f\ x))$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3x)\ f\ ((\lambda x.\ f^2x)\ x))$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3x)\ f\ (f^2x))$$

$$(\lambda x.\ (\lambda f\, x.\ f^3x)\ (f^2x))$$

3+2

$$(\lambda m\, n\, f\, x.\ m f (n f x))\ (\lambda f\, x.\ f^3 x)\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda n\, f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ (n f x))\ (\lambda f\, x.\ f^2 x)$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ ((\lambda f\, x.\ f^2 x)\ f\ x))$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ ((\lambda x.\ f^2 x)\ x))$$

$$(\lambda f\, x.\ (\lambda f\, x.\ f^3 x)\ f\ (f^2 x))$$

$$(\lambda x.\ (\lambda f\, x.\ f^3 x)\ (f^2 x))$$

$$(\lambda f\, x.\ f^3 (f^2 x))$$

3+2

$$(\lambda m\, n\, f\, x.\; m f (n f x))\; (\lambda f\, x.\; f^3 x)\; (\lambda f\, x.\; f^2 x)$$

$$(\lambda n\, f\, x.\; (\lambda f\, x.\; f^3 x)\; f\; (n f x))\; (\lambda f\, x.\; f^2 x)$$

$$(\lambda f\, x.\; (\lambda f\, x.\; f^3 x)\; f\; ((\lambda f\, x.\; f^2 x)\; f\; x))$$

$$(\lambda f\, x.\; (\lambda f\, x.\; f^3 x)\; f\; ((\lambda x.\; f^2 x)\; x))$$

$$(\lambda f\, x.\; (\lambda f\, x.\; f^3 x)\; f\; (f^2 x))$$

$$(\lambda x.\; (\lambda f\, x.\; f^3 x)\; (f^2 x))$$

$$(\lambda f\, x.\; f^3 (f^2 x)) = (\lambda f\, x.\; f^5 x)$$

Demo

Logic

- Formulae:

$$F \quad ::= \quad P \qquad \text{Prop. Variables}$$
$$\quad \mid \quad F \supset F \qquad \text{Implications}$$

- Inference Rules:

$$F \qquad \frac{\begin{matrix}[F_1] \\ \vdots \\ F_2\end{matrix}}{F_1 \supset F_2} \qquad \frac{\begin{matrix}\vdots \\ F_1 \supset F_2\end{matrix} \quad \begin{matrix}\vdots \\ F_1\end{matrix}}{F_2}$$

Logic

- Formulae:

$$F \quad ::= \quad P \qquad \text{Prop. Variables}$$
$$\quad | \quad F \supset F \qquad \text{Implications}$$

- Inference Rules:

$$F \qquad \frac{\begin{matrix}[F_1] \\ \vdots \\ F_2\end{matrix}}{F_1 \supset F_2} \qquad \frac{\begin{matrix}\vdots \\ F_1 \supset F_2\end{matrix} \quad \begin{matrix}\vdots \\ F_1\end{matrix}}{F_2}$$

$$F, \Gamma \vdash F \qquad \frac{F_1, \Gamma \vdash F_2}{\Gamma \vdash F_1 \supset F_2} \qquad \frac{\Gamma \vdash F_1 \supset F_2 \quad \Gamma \vdash F_1}{\Gamma \vdash F_2}$$

Logic

- Formulae:

$$F \;::=\; P \qquad \text{Prop. Variables}$$
$$\mid\; F \supset F \qquad \text{Implications}$$

- Inference Rules:

$$F \qquad \frac{\begin{array}{c}[F_1]\\ \vdots \\ F_2\end{array}}{F_1 \supset F_2} \qquad \frac{\begin{array}{cc}\vdots & \vdots\\ F_1 \supset F_2 & F_1\end{array}}{F_2}$$

$$F^x, \Gamma \vdash F \qquad \frac{F_1^x, \Gamma \vdash F_2}{\Gamma \vdash F_1 \supset F_2} \qquad \frac{\Gamma \vdash F_1 \supset F_2 \quad \Gamma \vdash F_1}{\Gamma \vdash F_2}$$

Correspondence

Inference rules

$$F^x, \Gamma \vdash F \qquad \frac{F_1^x, \Gamma \vdash F_2}{\Gamma \vdash F_1 \supset F_2}$$

$$\frac{\Gamma \vdash F_1 \supset F_2 \quad \Gamma \vdash F_1}{\Gamma \vdash F_2}$$

Typing rules

$$x : T, \Gamma \vdash x : T \qquad \frac{x : T_1, \Gamma \vdash M : T_2}{\Gamma \vdash \lambda x.M : T_1 \to T_2}$$

$$\frac{\Gamma \vdash M : T_1 \to T_2 \quad \Gamma \vdash N : T_1}{\Gamma \vdash M\,N : T_2}$$

Reduction

Beta-reduction

$$\dfrac{\dfrac{\overset{\vdots}{x : T_1, \Gamma \vdash M : T_2}}{\Gamma \vdash \lambda x.M : T_1 \to T_2} \quad \overset{\vdots}{\Gamma \vdash N : T_1}}{\Gamma \vdash (\lambda x.M)\, N : T_2}$$

$$\longrightarrow$$

$$\overset{\vdots}{\Gamma \vdash M[x := N] : T_2}$$

Reduction

Proof-normalisation (removal of detours)

$$\dfrac{\dfrac{\begin{array}{c}[F_1]\\ \vdots \\ F_2\end{array}}{F_1 \to F_2} \quad \begin{array}{c}\vdots\\ F_1\end{array}}{F_2}$$

$\longrightarrow$

$$\begin{array}{c}\vdots\\ F_1\\ \vdots\\ F_2\end{array}$$

Correspondence

Types	$\Leftrightarrow$	Formulae
Typed Terms	$\Leftrightarrow$	Proof
Evaluation	$\Leftrightarrow$	Proof Normalisation
Typing Problem	$\Leftrightarrow$	Finding a Proof
	$\vdots$	

- Program is correct by construction: take a proof, find the corresponding lambda-term (i.e. program), and finally evaluate term
- no problem with intuitionistic logic (for $\exists n.F$, an intuitionistic proof will construct such an n)

Classical Logic

- there are more classical proofs (and also more formulae provable)
- but classical logic is not constructive: $\exists a\, b$ such that a and b are irrational but a^b is rational.
- one can prove this without giving concrete values for a and b
- surprising result: classical proofs still correspond to programs

Raise and Handle

$$\frac{\Gamma \vdash M : T_2}{x^\circ : T_1 \to \bot, \Gamma \vdash \mathsf{raise}(x^\circ, M) : T_2}$$

$$\frac{x^\circ : T_1 \to \bot, \Gamma \vdash T_2 \quad x' : T_1, \Gamma \vdash N : T_2}{\Gamma \vdash \mathsf{let}\, x^\circ \,\mathsf{in}\, M \,\mathsf{handle}\, x^\circ\, x' \Rightarrow N : T_2}$$

$$\begin{array}{rcl} M\ (\mathsf{raise}(x^\circ, v')) & \longrightarrow & \mathsf{raise}(x^\circ, v') \\ (\mathsf{raise}(x^\circ, v))\ v' & \longrightarrow & \mathsf{raise}(x^\circ, v) \\ \mathsf{let}\, x^\circ \,\mathsf{in}\ v\ \mathsf{handle}\, x^\circ\, x' \Rightarrow N & \longrightarrow & v \end{array}$$

$$\begin{array}{l} \mathsf{let}\, x^\circ \,\mathsf{in}\ \mathsf{raise}(x^\circ, v)\ \mathsf{handle}\, x^\circ\, x' \Rightarrow N \\ \qquad \longrightarrow \quad N[x' := v] \end{array}$$

∀-Quantifier

- We can add the universal quantifier to the logic. What happens on the programming side?

$$\frac{\Gamma \vdash F \qquad X \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \forall X.F}$$

$$\frac{\Gamma \vdash \forall X.F_1}{\Gamma \vdash F_1[X := F_2]}$$

- Formulae: $F ::= X \mid F_1 \to F_2 \mid \forall X.F$

Data Types

- This will allow us to represent datatypes, such as
  - Booleans (either True or False)
  - Lists (either Nil or Cons)
  - Nats (either Zero or Successor)
  - Bin-trees (either Leaf or Node)
- The question is how to include them into the typing-system. Introducing them primitively is unsatisfactory. Why?

Syntax of PLC

- Types:

$$
\begin{array}{lcll}
T & ::= & X & \text{type variables} \\
 & \mid & T \to T & \text{function types} \\
 & \mid & \forall X.T & \forall\text{-type}
\end{array}
$$

- Terms:

$$
\begin{array}{lcll}
e & ::= & x & \text{variables} \\
 & \mid & e\ e & \text{applications} \\
 & \mid & \lambda x.e & \text{lambda-abstractions} \\
 & \mid & \Lambda X.e & \text{type-abstractions} \\
 & \mid & e\ T & \text{type-applications}
\end{array}
$$

Transitions in PLC

- We have the same transitions as in the lambda-calculus, e.g.

$$\overline{(\lambda x.e_1)e_2 \longrightarrow e_1[x := e_2]}$$

plus rules for type-abstractions and type-applications

$$\overline{(\Lambda X.e)T \longrightarrow e[X := T]}$$

- Confluence and termination holds for $\longrightarrow$.

Typing Rules

- Type-Generalisation

$$\frac{\Gamma \vdash e : T \quad X \notin \mathsf{ftv}(\Gamma)}{\Gamma \vdash \Lambda X.e : \forall X.T}$$

- Type-Specialisation

$$\frac{\Gamma \vdash e : \forall X.T_1}{\Gamma \vdash e\,T_2 : T_1[X := T_2]}$$

- Interestingly, for PLC the problems of type-checking and type-inference are computationally equivalent and **undecidable**!

Typing Rules

- Type-Generalisation

Therefore we explicitly annotate the type in lambda-abstractions

$$\lambda x : T.e$$

Type-checking is then trivial. (But is it useful?)

- Ty

$$\Gamma \vdash e\,T_2 : T_1[X := T_2]$$

- Interestingly, for PLC the problems of type-checking and type-inference are computationally equivalent and **undecidable**!

Datatypes

We are now returning to the question of representing datatypes in PLC.

- Booleans with values true and false is represented by

$$\text{bool} \stackrel{\text{def}}{=} \forall X.X \to (X \to X)$$

- true $\stackrel{\text{def}}{=} \Lambda X.\lambda x_1 : X.\lambda x_2 : X.x1$

  false $\stackrel{\text{def}}{=} \Lambda X.\lambda x_1 : X.\lambda x_2 : X.x2$

  These are the only two closed normal terms of type bool.

Lists

- Lists can be represented as

$$\boldsymbol{X}\ \mathsf{list} \stackrel{\mathsf{def}}{=} \forall \boldsymbol{Y}.\boldsymbol{Y} \to (\boldsymbol{X} \to \boldsymbol{Y} \to \boldsymbol{Y}) \to \boldsymbol{Y}$$

- $\mathsf{Nil} \stackrel{\mathsf{def}}{=} \boldsymbol{\Lambda X Y.\lambda x : Y.\lambda f : X \to Y \to Y.x}$

  $\mathsf{Cons} \stackrel{\mathsf{def}}{=} \ldots$

  These are infinitely closed normal terms of this type.

- We also have unit-, product- and sum-types. From this we can already build up all **algebraic types** (a.k.a. data types).

Possible Questions

- Question: A typed programming language is polymorphic if a term of the language may have different types (right or wrong)?
- PLC is at the heart of the immediate language in GHC: let-polymorphism of ML is compiled to (annotated) PLC.
- Describe the notion of beta-equality of terms in PLC. How can one decide that two typable PLC-terms are in this relation? Why does this fail for untypable terms?

Further Points

- Functional programming languages often allow bounds (constraints) on types: for example the membership functions of lists has type $X \to X$ list $\to$ bool, where X can only be a type with defined equality.
- Haskell generalises this idea by using type-classes.
- This is in contrast to object-oriented programming languages which use subtyping for modelling this.