Functional Programming
Theorems for Free!

Jevgeni Kabanov

Department of Computer Science
University of Tartu
Universal Types

Outline

Universal types introduce types as first-class language members:

- Type parametrization

\[ \text{double} = \Lambda X. \lambda f^{X \rightarrow X}. \lambda a^X. f (f \ a) \]

- Type application

\[ \text{double } [\text{Nat}] \ (\lambda x^{\text{Nat}}. (x + 2)) \ 1 \]

- Typing

\[ \text{double} : \forall X. (X \rightarrow X) \rightarrow X \rightarrow X \]

Lambda calculus with universal types is called System F. We will also use the notation \text{double}_{\text{Nat}} to denote a parametrized function.
Universal Types

Outline

- Universal types are more powerful than Hindley-Milner types
- However they cannot be inferred and need to be provided by the programmer
- This makes them less than comfortable in real life
- Haskell type system has System F extensions
- After type inference stage GHC translates every program to a simpler language based on System F
Deriving theorems

Example

Say \( r \) is a function of type

\[
r : \forall X. X^* \to X^*
\]

Where \( X^* \) is a list of \( X \)s. Then we can derive that for all types \( A \) and \( A' \) and for all total functions \( a : A \to A' \) we have:

\[
a^* \circ r_A = r_{A'} \circ a^*
\]

Where \( _* \) is equivalent to Haskell \( \text{map} :: (a \to b) \to [a] \to [b] \).
### Types as sets

**Definition**

We can interpret any type as a corresponding set:

- Nat is \( \{n \mid n \in 0\ldots\} \) and Bool = \{True, False\}.
- If \( A \) and \( B \) are types then \( A \times B = \{(a, b) \mid a \in A \land b \in B\} \).
- If \( A \) is a type then \( A^* \) is a set of lists with elements from \( A \).
- \( A \rightarrow B \) is a set of functions from \( A \) to \( B \).
- If \( X \) is a type variable and \( A(X) \) is a type dependent on \( X \) then the type \( \forall X. A(X) \) is a set of functions that take a set \( B \) and return an element of \( A(B) \).
Relations

Definition

Let’s recall relations:

- If $A$ and $A'$ are sets, we write $\mathcal{A} : A \sim A'$ to show that $\mathcal{A}$ is a relation between $A$ and $A'$, that is $\mathcal{A} \subseteq A \times A'$.
- We write $(x, y) \in \mathcal{A}$ if $x$ and $y$ are related by $\mathcal{A}$.
- Identity relation $\text{Id}_A : A \sim A$ is defined as $\text{Id}_A = \{(x, x) \mid x \in A\}$, in other words $(x, y) \in \text{Id}_A \equiv x = y$.
- Any function $a : A \to A'$ can be interpreted as a relation $a = \{(x, a\, x) \mid x \in A\}$, in other words $(x, x') \in a \equiv a\, x = x'$.
Identity and Cartesian

Definition

We can also interpret any type as a corresponding relation:

- Constant types are identity relations, $\text{Id}_\text{Bool} : \text{Bool} \sim \text{Bool}$, $\text{Id}_\text{Nat} : \text{Nat} \sim \text{Nat}$.

- For any relations $A : A \sim A'$ and $B : B \sim B'$ relation $A \times B : (A \times B) \sim (A' \times B')$ is defined as
  
  $$((x, y), (x', y')) \in A \times B \equiv (x, x') \in A \land (y, y') \in B$$

If $a$ and $b$ are functions then $(a \times b) (x, y) = (a x, b y)$.
Lists

**Definition**
For any relation $\mathcal{A} : A \sim A'$ the relation $\mathcal{A}^* : A^* \sim A'^*$ is defined as

$$(x_1, x_0^1; \ldots; x_n, x_0^n) \in \mathcal{A}^* \equiv (x_1, x_1^1) \in \mathcal{A} \land \ldots \land (x_n, x_n^n) \in \mathcal{A}$$

If $a$ is a function then $a^*$ is a map defined by

$a [x_1, \ldots, x_n] = [a x_1, \ldots, a x_n]$. 
Functions

Definition

For any relation \( A : A \sim A' \) and \( B : B \sim B' \) relation \( A \rightarrow B : (A \rightarrow B) \sim (A' \rightarrow B') \) is defined as

\[
(f, f') \in A \rightarrow B \equiv \forall (x, x') \in A : (f x, f' x') \in B
\]

If \( a \) and \( b \) are functions, then \( a \rightarrow b \) is not necessarily a function, but

\[
(f, f') \in a \rightarrow b \equiv a x = x' \land b (f x) = f' x'
\]

\[
\equiv b (f x) = f' (a x)
\]

\[
\equiv f' \circ a = b \circ f
\]
Universal types

**Definition**

Let $\mathcal{F}(\mathcal{X})$ be a relation depending on $\mathcal{X}$. The $\mathcal{F}$ corresponds to a function from relations to relations, so that for each $A : A \sim A'$ there exists $\mathcal{F}(A) : F(A) \sim F'(A')$. Then the relation $\forall \mathcal{X}. \mathcal{F}(\mathcal{X}) : \forall X. F(X) \sim \forall X'. F'(X')$ is defined as:

$$(g, g') \in \forall \mathcal{X}. \mathcal{F}(\mathcal{X}) \equiv \forall A : A \sim A', (g_A, g'_{A'}) \in \mathcal{F}(A)$$
Parametricity

**Theorem: Parametricity**

If $t$ is a closed term of type $T$, then $(t, t) \in \mathcal{T}$, where $\mathcal{T}$ is the relation corresponding to the type $T$. 
Rearrangement theorem

Derivation
Let \( r \) be a closed term of type \( \forall X. X^* \rightarrow X^* \). Parametricity gives that \( (r, r) \in \forall X. X^* \rightarrow X^* \). By definition of \( \forall \) on relations it is equivalent to

\[
\forall A : A \sim A', (r_A, r_{A'}) \in A^* \rightarrow A^*
\]

By definition of \( \rightarrow \) on relations:

\[
\forall A : A \sim A', \forall (xs, xs') \in A^*, (r_A \cdot xs, r_{A'} \cdot xs') \in A^*
\]

Let’s restrict \( A \)s to be functions \( a : A \rightarrow A' \) and specialize:

\[
\forall a \forall xs : xs' = a^* \cdot xs \Rightarrow a^* (r_A \cdot xs) = r_{A'} \cdot xs'
\]

\[
\equiv \forall a : a^* (r_A \cdot xs) = r_{A'} (a^* \cdot xs)
\]

\[
\equiv \forall a : a^* \circ r_A = r_{A'} \circ a^*
\]
Map theorem

Derivation

Let $m$ be a closed term of type $\forall X.\forall Y.(X \rightarrow Y) \rightarrow (X^* \rightarrow Y^*)$. Parametricity gives that $(m, m) \in \forall X.\forall Y.(X \rightarrow Y) \rightarrow (X^* \rightarrow Y^*)$. Taking $X = a, Y = b$ and applying definition of $\forall$ twice we get:

$$\forall a \forall b : (m_{AB}, m_{A'B'}) \in (a \rightarrow b) \rightarrow (a^* \rightarrow b^*)$$

Further applying the definition of $\rightarrow$:

$$\forall a \forall b \forall (f, f') \in (a \rightarrow b) : (m_{AB} f, m_{A'B'} f') \in (a^* \rightarrow b^*)$$

$$\equiv \forall a \forall b : f' \circ a = b \circ f \Rightarrow (m_{AB} f, m_{A'B'} f') \in (a^* \rightarrow b^*)$$

$$\equiv \forall a \forall b : f' \circ a = b \circ f \Rightarrow m_{A'B'} f' \circ a^* = b^* \circ m_{AB} f$$
Map corollary

Derivation

Out previous result was for all \(a\) and \(b\)

\[
   f' \circ a = b \circ f \Rightarrow m_{A'B'} f' \circ a^* = b^* \circ m_{AB} f
\]

Taking \(A' = B' = B\), \(b = f' = \text{Id}_B\), \(a = f\) we get

\[
   \text{Id}_B \circ f = \text{Id}_B \circ f \Rightarrow m_{BB}(\text{Id}_B) \circ f^* = (\text{Id}_B)^* \circ m_{AB}(f)
\]

The premiss is obviously a tautology and since \((\text{Id}_B)^* = \text{Id}_{B^*}\)
the result can be rewritten as

\[
   m_{AB}(f) = m_{BB}(\text{Id}_B) \circ f^*
\]

Which means that any function \(m\) of type
\[
   \forall X. \forall Y. (X \rightarrow Y) \rightarrow (X^* \rightarrow Y^*)
\]
is equivalent to map up to element rearrangement.
Sort and Dup theorem

**Derivation**

Let $s$ be of type $\forall X. (X \to X \to \text{Bool}) \to (X^* \to X^*)$ (examples are sort that sorts the list after a ordering and dup that removes adjacent duplicates after equivalence). Then for all $a$:

$$\forall x, y \in A : x \prec y = a \ x \prec^\prime a \ y \Rightarrow a^\ast \circ s_A (\prec) = s_{A'} (\prec') \circ a^\ast$$

For **sort** this means that map commutes with sort if $f$ preserves ordering:

$$\forall x, y \in A : x \prec y = a \ x \prec^\prime a \ y \Rightarrow a^\ast \circ s_A (\prec) = s_{A'} (\prec') \circ a^\ast$$

For **dup** this means that map commutes with dup if $f$ preserves equivalence:

$$\forall x, y \in A : x \equiv y = a \ x \equiv^\prime a \ y \Rightarrow a^\ast \circ s_A (\equiv) = s_{A'} (\equiv') \circ a^\ast$$
Fold theorem

Derivation

fold type is $\forall X. \forall Y. (X \to Y \to Y) \to Y \to X^* \to Y$. Applying the definition of $\forall$ twice and specializing to functions $a : A \to A'$, $b : B \to B'$:

$$(\text{fold}_{AB}, \text{fold}_{A'B'}) \in (a \to b \to b) \to b \to a^* \to b$$

Applying definition of $\to$ twice we get that for all $(\oplus, \oplus') \in (a \to b \to b)$:

$$u' = b \ u \Rightarrow (\text{fold}_{AB} (\oplus) u, \text{fold}_{A'B'} (\oplus') u') \in a^* \to b$$

Whereas $\forall (\oplus, \oplus') \in (a \to b \to b)$ can be interpreted as

$$\forall x \in A, \forall y \in B : b (x \oplus y) = (a \ x) \oplus' (b \ y)$$
Fold theorem

Derivation

The resulting theorem for fold looks like:

\[ \forall x \in A, \forall y \in B : b(x \oplus y) = (a x) \oplus' (b y) \land u' = b u \]
\[ \implies b \circ \text{fold}_{AB}(\oplus) u = \text{fold}_{A'B'}(\oplus') u' \circ a^* \]

Although it seems complicated, it states that if \( a \) and \( b \) provide a homomorphism between algebra structures \((A, B, \oplus, u)\) and \((A', B', \oplus', u')\) then \( a^* \) and \( b \) provide a homomorphism between algebra structures \((A^*, B, \text{fold}_{AB}(\oplus)u)\) and \((A', B', \text{fold}_{A'B'}(\oplus')u')\).

Similarly to map we can prove that every function \( f \) of fold type can be expressed as:

\[ f_{AB} c n = \text{fold}_{AB} c n \circ f_{A^*} \text{cons}_A \text{nil}_A \]
Finally

Outline

- Parametricity breaks in the presence of fixpoint combinator, it needs additionally for qualified functions to be strict ($f \bot = \bot$). Since Haskell provides recursive definitions this must be taken into account.

- Since every polymorphic type gives rise to a theorem, this approach can yield a lot more results, though most of them are less useful.

- It can also help to make steps in some more powerful theorem, only requiring parametricity (e.g. Hindley/Milner to Girard/Reynolds type system isomorphism).

- Theorems can be (and are) generated completely automatically! Try http://haskell.as9x.info/cgi-bin/ftonline.pl