Programming = proving?

The Curry-Howard correspondence today

Sixth lecture

Theorems for free: parametricity and logical relations

Xavier Leroy

Collège de France

2018-12-19

COLLÈGE
DE FRANCE
1530

Quiz

In the polymorphic lambda-calculus (system F), which terms have the following types?

$\forall X.\ X$

$\forall X.\ X \to X$

$\forall X.\ X \to \texttt{int}$

$\forall X.\ X \to X \to X$

Quiz

In the polymorphic lambda-calculus (system F), which terms have the following types?

$\forall X.\ X$ none (logical inconsistency otherwise!)

$\forall X.\ X \to X$

$\forall X.\ X \to \texttt{int}$

$\forall X.\ X \to X \to X$

Quiz

In the polymorphic lambda-calculus (system F), which terms have the following types?

$\forall X.\, X$	none (logical inconsistency otherwise!)
$\forall X.\, X \to X$	identity $\Lambda X.\lambda x.\, x$; others?
$\forall X.\, X \to \texttt{int}$	
$\forall X.\, X \to X \to X$	

Quiz

In the polymorphic lambda-calculus (system F), which terms have the following types?

$\forall X.\ X$	none (logical inconsistency otherwise!)
$\forall X.\ X \to X$	identity $\Lambda X.\lambda x.\ x$; others?
$\forall X.\ X \to \texttt{int}$	constant functions $\Lambda X.\lambda x.\ n$; others?
$\forall X.\ X \to X \to X$	

Quiz

In the polymorphic lambda-calculus (system F), which terms have the following types?

$\forall X.\ X$	none (logical inconsistency otherwise!)
$\forall X.\ X \to X$	identity $\Lambda X.\lambda x.\ x$; others?
$\forall X.\ X \to \texttt{int}$	constant functions $\Lambda X.\lambda x.\ n$; others?
$\forall X.\ X \to X \to X$	Church's Booleans, $\Lambda X.\lambda x.\lambda y.\ x$ and $\Lambda X.\lambda x.\lambda y.\ y$; others?

Quiz

In the polymorphic lambda-calculus (system F), which terms have the following types?

$\forall X.\, X$	none (logical inconsistency otherwise!)
$\forall X.\, X \to X$	identity $\Lambda X.\lambda x.\, x$; others?
$\forall X.\, X \to \texttt{int}$	constant functions $\Lambda X.\lambda x.\, n$; others?
$\forall X.\, X \to X \to X$	Church's Booleans, $\Lambda X.\lambda x.\lambda y.\, x$ and $\Lambda X.\lambda x.\lambda y.\, y$; others?

How can we show that there are no other terms (closed and in normal form) of these types?

Theorems about lists

(Phil Wadler, *Theorems for free!*, 1991)

In system F extended with the type t^* of lists of t, the following equations always hold:

$$\mathtt{map}\, f\, (F\, x) = F\, (\mathtt{map}\, f\, x) \qquad \text{if } F : \forall X.\, X^* \to X^*$$

$$\mathtt{map}\, f\, (G\, x\, y) = G\, (\mathtt{map}\, f\, x)\, (\mathtt{map}\, f\, y) \qquad \text{if } G : \forall X.\, X^* \to X^* \to X^*$$

$$\mathtt{map}\, f\, (H\, x) = H\, (\mathtt{map}\, (\mathtt{map}\, f)\, x) \qquad \text{if } H : \forall X.\, X^{**} \to X^*$$

$$\mathtt{map}\, f\, (\Phi\, (p \circ f)\, x) = \Phi\, p\, (\mathtt{map}\, f\, x) \qquad \text{if } \Phi : \forall X.\, (X \to \mathtt{bool}) \to X^* \to X^*$$

where $f : A \to B$, $p : B \to \mathtt{bool}$, and $\mathtt{map}\, f\, [x_1; \ldots; x_n] = [f\, x_1; \ldots; f\, x_n]$.

We can prove these equations for $F = \mathtt{rev}$ or $G = \mathtt{append}$ or $H = \mathtt{concat}$ or $\Phi = \mathtt{filter}$; but why do they hold for all functions having these polymorphic types?

Teaching complex numbers

(A fable told by John C. Reynolds)

Two teachers, two sections. During the first lecture:

- Prof. Descartes: "$\mathbb{C} = \mathbb{R} \times \mathbb{R}$". (Cartesian representation)
  Defines the injection $\mathbb{R} \to \mathbb{C}$, then i, $+$, $\times$, $^{-1}$, $\bar{z}$, $|z|$.
- Prof. Bessel: "$\mathbb{C} = \mathbb{R}^+ \times \mathbb{R}$". (polar representation)
  Defines the injection $\mathbb{R} \to \mathbb{C}$, then i, $+$, $\times$, $^{-1}$, $\bar{z}$, $|z|$.

At the next lecture, the two sections are interchanged.

Neither Descartes nor Bessel commit any mathematical error, even though they are judged (by the students) based on the other's definitions.

Abstract types

What prevents Descartes and Bessel from making mistakes is that, starting from the second lecture, they treat complex numbers as an abstract type: a type name $\mathbb{C}$ and operations over this type.

Both refrain from examining the concrete representation of complex numbers, e.g. to project the first component of a number, which would lead to contradictions:

$$\texttt{proj}_1(i) = \texttt{proj}_1((0,1)) = 0 \qquad \text{for Descartes}$$
$$\texttt{proj}_1(i) = \texttt{proj}_1((1,\pi/2)) = 1 \qquad \text{for Bessel}$$

The moral of the fable:

> *Type structure is a syntactic discipline for enforcing levels of abstraction.*
>
> *(John C. Reynolds, 1983)*

Type abstraction and polymorphism

As observed by Reynolds (1974), one way to treat a type abstractly is to make its users polymorphic over the name of the type.

In the example of complex numbers, the students are polymorphic functions over the type $\mathbb{C}$:

$$\begin{aligned}\text{student} : \forall\mathbb{C}.\, \{\, &\texttt{inj} : \mathbb{R} \to \mathbb{C};\;\; \texttt{i} : \mathbb{C};\\ &\texttt{conj} : \mathbb{C} \to \mathbb{C}; \texttt{module} : \mathbb{C} \to \mathbb{R}^{+};\\ &\texttt{add} : \mathbb{C} \to \mathbb{C} \to \mathbb{C};\;\; \texttt{mul} : \mathbb{C} \to \mathbb{C} \to \mathbb{C}\,\} \to t\end{aligned}$$

The type $\mathbb{C}$ being a type variable, the only way to build and use values of type $\mathbb{C}$ is through the functions passed as argument.

Parametricity

An intuition: polymorphism is parametric, that is, two instantiations $f[A]$ and $f[B]$ of a polymorphic function $f : \forall X \ldots$ over two different types implement the same algorithm.

A dual intuition: abstract types are independent of their representations, that is, two implementations of an abstract type can be distinguished only through the constants and operations provided over this type.

A proof technique: logical relations
($\approx$ interpreting types by relations between values).

Various applications: "theorems for free" (true for all functions having a given type), non-inhabitation results, isomorphisms between functional encodings and algebraic types, etc.

I

Logical relations

Definability and logical relations (Plotkin)

G. Plotkin, *λ-definability and Logical Relations*, 1973;
G. Plotkin, *λ-definability in the full type hierarchy*, 1980.

Plotkin wanted to characterize the functions that can be defined in the pure lambda-calculus (1973) or the simply-typed lambda calculus (1980) possibly extended with constants.

For instance: in the simply-typed case, we assume given a base type o that is interpreted by the set $O = \{T, F\}$.

$$[\![o]\!] = O$$
$$[\![t_1 \to t_2]\!] = [\![t_1]\!] \to [\![t_2]\!] \quad \text{(sets of functions)}$$

What are the set-theoretic functions $O \to O$ definable by lambda-terms?
without constants: identity $T \mapsto T,\ F \mapsto F$
with $\mathtt{T}$ and $\mathtt{F}$ constants: all functions except negation $T \mapsto F,\ F \mapsto T$

Logical relations

Plotkin's idea: definable functions satisfy some relations, those that are compatible with function abstraction and application. Plotkin called them logical relations.

Definition: a n-place logical relation is a family of relations $R^t \subseteq [\![t]\!] \times \cdots \times [\![t]\!]$ indexed by a type t, such that

$$R^{t \to s}\,(f_1, \ldots, f_n) \iff \forall x_1, \ldots, x_n,\ R^t\,(x_1, \ldots, x_n) \Rightarrow R^s\,(f_1\,x_1, \ldots, f_n\,x_n)$$

In other words: functions are related if and only if they map related arguments to related results.

Remark: the logical relation is entirely determined by the relations R^o over the base types o.

Logical relations

The only cases used in practice are $n = 1$ and $n = 2$.

Unary logical relation: a predicate over O that "hereditarily" extends to functions

$$R^o(x) \qquad \text{chosen freely}$$

$$R^{t \to s}\,(f) \stackrel{def}{=} \forall x,\ R^t\,(x) \Rightarrow R^s\,(f\ x)$$

Binary logical relation: (or just "logical relation")

$$R^o(x_1, x_2) \qquad \text{chosen freely}$$

$$R^{t \to s}\,(f_1, f_2) \stackrel{def}{=} \forall x_1, x_2,\ R^t\,(x_1, x_2) \Rightarrow R^s\,(f_1\ x_1, f_2\ x_2)$$

The fundamental theorem of logical relations

An interpretation of terms:

$$\begin{aligned} [\![x]\!]\ \rho &= \rho(x) \\ [\![\lambda x.\, M]\!]\ \rho &= v \mapsto [\![M]\!]\ (\rho + (x \mapsto v)) \\ [\![M\ N]\!]\ \rho &= [\![M]\!]\ \rho\ ([\![N]\!]\ \rho) \end{aligned}$$

If a term is well typed, its interpretations in related environments are related.

Theorem

If $\Gamma \vdash M : t$ *and* $R^{\Gamma(x)}\ (\rho_1(x), \rho_2(x))$ *for all* $x \in Dom(\Gamma)$, *then* $R^t\ ([\![M]\!]\ \rho_1,\ [\![M]\!]\ \rho_2)$.

Corollary

If $\vdash M : t$ *then* $R^t([\![M]\!], [\![M]\!])$.

Back to the definability problem

Any definable function $f : O \to O$ (that is, of the form $[\![M]\!]$ for some term M) must therefore be related to itself at type $o \to o$:

$$R^{o \to o}\ (f, f) \quad \text{that is,} \quad \forall x_1, x_2,\ R^o(x_1, x_2) \Rightarrow R^o(f\ x_1, f\ x_2)$$

Taking $R^o = \{(T, F)\}$, we see that the following functions are not definable:

- the constant function $f(x) = T$ since $(f(T), f(F)) = (T, T) \notin R^o$
- the constant function $f(x) = F$ since $(f(T), f(F)) = (F, F) \notin R^o$
- negation $f(T) = F,\ f(F) = T$ since $(f(T), f(F)) = (F, T) \notin R^o$

Syntactic logical relations

(R. Statman, *Logical relations and the typed λ-calculus*, Inf&Control 65, 1985)

Statman reformulated logical relations without using a set-theoretic model of typed lambda-calculus, just as relations between (syntactic) lambda-terms modulo β-equivalence:

$$R^o\ (M_1, \ldots, M_n) = \quad \text{chosen as we wish}$$

$$R^{t \to s}\ (M_1, \ldots, M_n) \stackrel{def}{=} \forall N_1, \ldots, N_n,\ R^t\ (N_1, \ldots, N_n) \Rightarrow R^s\ (M_1\ N_1, \ldots, M_n\ N_n)$$

It is Statman who gave the name "fundamental theorems of logical relations" to the following result:

Theorem

If $\vdash M : t$, then $R^t(M, \ldots, M)$ for all logical relations R.

Connection with strong normalization

Statman remarked that Tait's proof (1967) of strong normalization for simply-typed lambda-calculus is an instance of unary logical relation.

Define the set $RED(t)$ of reducible terms of type t by induction over t:

$$RED(o) = \{M \mid M \text{ is strongly normalizing}\}$$
$$RED(t \to s) = \{M \mid \forall N \in RED(t), M\,N \in RED(s)\}$$

We then show:

1. If $\vdash M : t$ then $M \in RED(t)$.
2. If $M \in RED(t)$ then M is strongly normalizing.

RED is a unary logical relation, and (1) follows from the fundamental theorem of logical relations.

Extension to abstract types

(John C. Reynolds, *Types, abstraction and parametric polymorphism*, 1983)

Reynolds observed that logical relations enable us to reason about an abstract type and its different implementations, provided a type name can be interpreted by different sets.

Continuing the example of Cartesian and polar numbers:

$$\begin{aligned} [\![R]\!]_1 &= \mathbb{R} & [\![R]\!]_2 &= \mathbb{R} \\ [\![C]\!]_1 &= \mathbb{R} \times \mathbb{R} & [\![C]\!]_2 &= \mathbb{R}^+ \times \mathbb{R} \\ [\![t \to s]\!]_1 &= [\![t]\!]_1 \to [\![s]\!]_1 & [\![t \to s]\!]_2 &= [\![t]\!]_2 \to [\![s]\!]_2 \end{aligned}$$

A (binary) logical relation is, then, a family $R^t \subseteq [\![t]\!]_1 \times [\![t]\!]_2$ such that

$$R^{t \to s}\,(f_1, f_2) \stackrel{def}{=} \forall x_1 \in [\![t]\!]_1, x_2 \in [\![t]\!]_2,\ R^t\,(x_1, x_2) \Rightarrow R^s\,(f_1\,x_1, f_2\,x_2)$$

Extension to abstract types

The fundamental theorem (called the abstraction theorem by Reynolds) still holds: the interpretations of a well-typed term in related environments are related.

Theorem

If $\Gamma \vdash M : t$ *and* $R^{\Gamma(x)}\ (\rho_1(x), \rho_2(x))$ *for all* $x \in Dom(\Gamma)$*, then* $R^t\ (\llbracket M \rrbracket\ \rho_1,\ \llbracket M \rrbracket\ \rho_2)$.

Representation independence

Application to complex numbers: take Γ to be the signature of complex operations, ρ_1 their implementation with Cartesian representation, ρ_2 their implementation with polar representation.

$$\Gamma = \mathtt{inj} : R \to C;\ \mathtt{i} : C;\ \mathtt{conj} : C \to C; \ldots$$
$$\rho_1 = \{\mathtt{inj} \mapsto \lambda x.\,(x, 0);\ \mathtt{i} \mapsto (0, 1);\ \mathtt{conj} \mapsto \lambda(x, y).\,(x, -y); \ldots\}$$
$$\rho_2 = \{\mathtt{inj} \mapsto \lambda x.\,(x, 0);\ \mathtt{i} \mapsto (1, \pi/2);\ \mathtt{conj} \mapsto \lambda(r, \theta).\,(r, -\theta); \ldots\}$$

Concerning logical relations at base types, we take

$$R^R\,(x_1, x_2) = (x_1 = x_2)$$
$$R^C\,((x, y), (r, \theta)) = (x = r\cos\theta \wedge y = r\sin\theta)$$

Representation independence

Then, it suffices to show that the operations in the two implementations are pairwise related:

$$R^{\Gamma(op)}\ (\rho_1(op), \rho_2(op)) \quad \text{for all } op \in Dom(\Gamma)$$

The fundamental theorem, then, guarantees that all computations of type R give the same results with both implementations of complex numbers:

$$\Gamma \vdash M : R \Longrightarrow [\![M]\!]\ \rho_1 = [\![M]\!]\ \rho_2$$

More generally: two implementations of an abstract type that are related by a logical relation are observationally equivalent, even if their representation types differ.

Extension to polymorphism (system F)

(John C. Reynolds, *Types, abstraction and parametric polymorphism*, 1983)

At the end of his 1983 paper, Reynolds tries to extend logical relations to the polymorphic lambda-calculus (system F).

Problem: impredicativity. A term with polymorphic type $\forall X.\, t$ can be instantiated as $t\{X \leftarrow t'\}$ for any type t', including $t' = \forall X.\, t$. Example:

$$\text{if} \quad id : \forall X.\, X \to X \quad \text{then} \quad id\ [\forall X.\, X \to X]\ id : \forall X.\, X \to X$$

It is therefore impossible to model $\forall X.\, t$ as the intersection of all instantiations:

✘ $$[\![\forall X.\, t]\!] = \bigcap_{t' \text{ type}} [\![t\{X \leftarrow t'\}]\!]$$

✘ $$R^{\forall X.\, t}(x_1, x_2) = \forall t',\ R^{t\{X \leftarrow t'\}}(x_1, x_2)$$

Strong normalization of system F

Girard (1972) ran into a similar problem when proving strong normalization for system F. A naive extension of the reducibility method leads to a circular definition:

$$\text{✘} \quad RED(\forall X.\, t) = \{\, M \mid \forall t',\ M[t'] \in RED(t\{X \leftarrow t'\}) \,\}$$

Girard's idea is to interpret type variables X not just by the sets $RED(t')$ for every type t', but by a larger class of sets: the reducibility candidates (*candidats de réductibilité*).

A set U of terms is a reducibility candidate if:

1. every $M \in U$ is strongly normalizing;
2. U is closed under β-expansion: if $M \to_\beta M'$ and $M' \in U$ then $M \in U$
3. U is closed under some β-reductions.
   (See Girard, Lafont, Taylor, *Proofs and Types*, ch. 14)

Strong normalization of system F

Reductibility: (Φ : type variable $\to$ candidate)

$$
\begin{aligned}
RED(o, \Phi) &= \{M \mid M \text{ is strongly normalizing}\} \\
RED(t \to s, \Phi) &= \{M \mid \forall N \in RED(t, \Phi), M\ N \in RED(s, \Phi)\} \\
RED(X, \Phi) &= \Phi(X) \\
RED(\forall X.\, t, \Phi) &= \{M \mid \forall t', \forall R \in \text{candidates}(t'), M[t'] \in RED(t,\ \Phi + X \mapsto R\}
\end{aligned}
$$

We then show:

1. If $M \in RED(t, \Phi)$ then M is strongly normalizing.
2. $RED(t, \Phi)$ is a reducibility candidate.
3. If $\vdash M : t$ then $M \in RED(t, \Phi)$.

Logical relations for system F

The same idea applies to logical relations: we must be able to interpret type variables X by "semantic" relations taken from a larger set than the set of "syntactic" relations, $\{R^t \mid t \text{ type}\}$.

$$
\begin{aligned}
R^o_\Phi(x_1, x_2) &\qquad \text{chosen as we wish} \\
R^{t\to s}_\Phi\,(f_1, f_2) &\stackrel{def}{=} \forall x_1, x_2,\ R^t_\Phi\,(x_1, x_2) \Rightarrow R^s_\Phi\,(f_1\,x_1, f_2\,x_2) \\
R^X_\Phi\,(x_1, x_2) &\stackrel{def}{=} \Phi(X)\,(x_1, x_2) \\
R^{\forall X.\,t}_\Phi\,(x_1, x_2) &\stackrel{def}{=} \forall U \in \mathcal{U},\ R^t_{\Phi + X\mapsto U}\,(x_1, x_2)
\end{aligned}
$$

It remains to (1) build a model for the types and the terms of system F, and (2) define the set $\mathcal{U}$ of "semantic" relations.

Models for system F

Set-theoretic models, for instance Reynolds (1983):
cannot exist for reasons of cardinality.
(Reynolds, *Polymorphism is not set-theoretic*, 1984).

Models using Scott domains:
for instance, Bruce, Meyer, Mitchell, 1990, used by Wadler, 1991.
The relations $\mathcal{U}$ must be admissible (closed under limits).

Categorical models:
for instance, Girard's coherent spaces.

Purely syntactic approach in the style of Statman,
for instance Harper, *Practical foundations for prog. lang.* chap. 48.
The relations $\mathcal{U}$ must be closed under β-expansion and observational equivalence.

Embedding system F and the relations in a type theory.
See part III.

II

Theorems for free
and other applications

The type $\forall X.\, X$ is empty

Assume $\vdash M : \forall X.\, X$.

By the fundamental theorem, $R^{\forall X.\, X}\ (\llbracket M \rrbracket, \llbracket M \rrbracket)$.

Let's interpret X by the empty relation $\emptyset$. We have:

$$R^{X}_{X \mapsto \emptyset}\ (\llbracket M \rrbracket, \llbracket M \rrbracket) \qquad \text{that is,} \qquad (\llbracket M \rrbracket, \llbracket M \rrbracket) \in \emptyset$$

This is impossible. Hence, M does not exist.

Values of type $\forall X.\, X \to X$

Assume $\vdash M : \forall X.\, X \to X$.

Let t be a type and $x \in [\![t]\!]$. We show that $[\![M]\!]\, x = x$.

By the fundamental theorem: $R^{\forall X.\, X \to X}\, ([\![M]\!], [\![M]\!])$.

Interpreting X by the relation $\overline{X} = \{(x, x)\}$, we get:

$$\forall y_1, y_2,\ (y_1, y_2) \in \overline{X} \Rightarrow ([\![M]\!]\, y_1, [\![M]\!]\, y_2) \in \overline{X}$$

We take $y_1 = y_2 = x$ and obtain $[\![M]\!]\, x = x$.

This holds for any x. Hence, $[\![M]\!]$ is the identity function.

In some models (but not all models) it follows that $M =_{\beta\eta} \Lambda X.\lambda x : X.\, x$.

Values of type $\forall X.\, X \to X \to X$

Assume given a base type `bool` with two elements T and F.

Assume $\vdash M : \forall X.\, X \to X \to X$. Let t be a type and $x, y \in [\![t]\!]$.

We interpret X by the relation $\overline{X} \subseteq [\![t]\!] \times [\![bool]\!] = \{\, (x, T); (y, F) \,\}$.

As a consequence of the fundamental theorem, we get:

$$\forall u_1, u_2, v_1, v_2.\, (u_1, u_2) \in \overline{X} \wedge (v_1, v_2) \in \overline{X} \Rightarrow ([\![M]\!]\ u_1\ v_1, [\![M]\!]\ u_2\ v_2) \in \overline{X}$$

We take $u_2 = T$ and $v_2 = F$, thus forcing $u_1 = x$ and $v_1 = y$, and therefore

$$([\![M]\!]\ x\ y, [\![M]\!]\ T\ F) \in \overline{X}$$

If $[\![M]\!]\ T\ F = T$ then $[\![M]\!]\ x\ y = x$, for any x and y.
If $[\![M]\!]\ T\ F = F$ then $[\![M]\!]\ x\ y = y$, for any x and y.

Functional relations

If $f : A \to B$ is a (set-theoretic) function, we can interpret a type variable X by its graph $\overline{f} \stackrel{def}{=} \{(a, b) \mid b = f(a)\}$.

Two elements x, y are related at type X iff $y = f(x)$.

Two functions g, h are related at type $X \to X$ iff $\forall x,\ h(f\ x) = f(g\ x)$.

Two functions g, h are related at type $X \to X \to X$ iff
$\forall x, y,\ h\ (f\ x)\ (f\ y) = f(g\ x\ y)$.

Values of type $\forall X.\, X \to X \to X$ (alternate approach)

Assume $\vdash M : \forall X.\, X \to X \to X$.

Interpreting X by the graph of a function $\bar{f}$, we get

$$\forall x, y,\ [\![M]\!]\ (f\ x)\ (f\ y) = f\ ([\![M]\!]\ x\ y)$$

Let t be a type, and $x, y \in [\![t]\!]$.

Define $f : \texttt{bool} \to [\![t]\!]$ as $f(T) = x$ and $f(F) = y$.

It follows

$$[\![M]\!]\ x\ y = f([\![M]\!]\ T\ F) = \begin{cases} x & \text{if } [\![M]\!]\ T\ F = T \\ y & \text{if } [\![M]\!]\ T\ F = F \end{cases}$$

Extensions: products, sums, lists

It is easy to define logical relations for product, sum, and list types:

$$R^{t \times s} = \{((x, y), (x', y')) \mid (x, x') \in R^t \wedge (y, y') \in R^s\}$$

$$R^{t+s} = \{(\mathtt{inj}_1(x), \mathtt{inj}_1(x') \mid (x, x') \in R^t\} \cup \{(\mathtt{inj}_2(y), \mathtt{inj}_1(y') \mid (y, y') \in R^s\}$$

$$R^{t^*} = \{([x_1, \ldots, x_n],\ [x'_1, \ldots, x'_n] \mid (x_i, x'_i) \in R^t \text{ for } i = 1, \ldots, n\}$$

Theorems about lists

(Phil Wadler, *Theorems for free!*, 1991)

Wadler's "theorems for free" about lists follow easily from the fundamental theorem, with the help of functional relations.

$$\mathtt{map}\, f\, (\llbracket F \rrbracket\, x) = \llbracket F \rrbracket\, (\mathtt{map}\, f\, x) \qquad \text{if } F : \forall X.\, X^* \to X^*$$

$$\mathtt{map}\, f\, (\llbracket G \rrbracket\, x\, y) = \llbracket G \rrbracket\, (\mathtt{map}\, f\, x)\, (\mathtt{map}\, f\, y) \qquad \text{if } G : \forall X.\, X^* \to X^* \to X^*$$

$$\mathtt{map}\, f\, (\llbracket H \rrbracket\, x) = \llbracket H \rrbracket\, (\mathtt{map}\, (\mathtt{map}\, f)\, x) \qquad \text{if } H : \forall X.\, X^{**} \to X^*$$

$$\mathtt{map}\, f\, (\llbracket \Phi \rrbracket\, (p \circ f)\, x) = \llbracket \Phi \rrbracket\, p\, (\mathtt{map}\, f\, x) \qquad \text{if } \Phi : \forall X.\, (X \to \mathtt{bool}) \to X^* \to X^*$$

Note: if X is interpreted by the functional relation $\overline{f}$,
then the relation at type X^* is $\overline{\mathtt{map}\, f}$,
and the relation at type X^{**} is $\overline{\mathtt{map}\, (\mathtt{map}\, f)}$.

Values of type $\forall X.\, X \to (X \to X) \to X$

Let M be a term such that $\vdash M : \forall X.\, X \to (X \to X) \to X$.

We assume given a base type $\mathtt{nat}$ interpreted by $\mathbb{N}$.

Let t be a type, $f : [\![t]\!] \to [\![t]\!]$ a function over t, and $a \in [\![t]\!]$.

We interpret type X by the relation

$$\overline{X} = \{(0, a);\ (1, f\ a); \ldots; (n, f^n\ a); \ldots\}$$

Function *succ* and function f are related at type $X \to X$:
if $(n, x) \in \overline{X}$, then $x = f^n\ a$, hence $(succ\ n,\ f\ x) = (n+1, f^{n+1}\ a) \in \overline{X}$.

Consequently, $[\![M]\!]\ 0\ succ$ and $[\![M]\!]\ a\ f$ are related at type X.

Hence $[\![M]\!]\ a\ f = f^n\ a$ where n is determined by $n = [\![M]\!]\ 0\ succ$.

M is therefore the n-th Church integer.

Functional encodings of inductive types

We saw earlier (lecture of Nov 28th) that any inductive type, not just natural numbers, can be encoded in system F as polymorphic functions $\forall X. \ldots \to X$.

The previous example shows that the functional encoding of natural numbers is isomorphic to natural numbers, in the sense that all the values of this type are images of numbers by the encoding.

This result extends to all functional encodings of all inductive types.
(Abadi, Plotkin, *A logic for parametric polymorphism*, 1993.)

Exercise: show that $\forall X.\ (A \to B \to X) \to X$ is isomorphic to $A \times B$.

Parametricity and higher-order abstract syntax

To represent terms containing variables and binders:

- First-order abstract syntax:
  explicit representation of variables.
- Higher-order abstract syntax:
  use the “lambda” of the host language.

Example: pure lambda-calculus.

```
type lam =
  | Var of int
  | Lam of lam
  | App of lam * lam
```

First order
(de Bruijn indices)

```
type lam =

  | Lam of (lam -> lam)
  | App of lam * lam
```

Higher order

Parametricity and higher-order syntax

```
type lam = Lam of (lam -> lam) | App of lam * lam
```

Problem: in a language like OCaml, there are many expressions of type `lam` that do not represent a lambda-term, such as

```
Lam (fun x -> Lam (fun y -> match x with Lam _ -> x | App _ -> y))
```

Washburn and Weirich (2008) conjectured that the functional encoding of the algebraic type above,

$$\forall X.\ ((X \to X) \to X) \to (X \to X \to X) \to X$$

does not contain such "exotic" terms, because of parametricity with respect to X.

Parametricity and higher-order syntax

R. Atkey, *Syntax for free*, TLCA 2009.

```
type lam =
  | Var of int
  | Lam of lam
  | App of lam * lam
```

Atkey (2009) proved that the type

$$\forall X.\, ((X \to X) \to X) \to (X \to X \to X) \to X$$

is isomorphic to closed terms represented with de Bruijn indices, that is, to the subset of the algebraic type `lam` above where there are no free variables.

(The proof uses Kripke logical relations, an extension of logical relations with worlds.)

III

Parametricity in type theory

Which logic to talk about parametricity?

Until now, we have described parametricity using standard mathematical logic, via a model $[\![\cdots]\!]$ that interprets types and terms of the language (system F) into this logic.

Another possibility: develop a "bespoke" logic to reason about terms and logical relations of system F. The fundamental theorem of logical relations is an axiom of this logic.
(Abadi and Plotkin, *A logic for parametric polymorphism*. TLCA 1993.)

Third possibility: use a unified formalism (type theory, Pure Type System) capable of describing simultaneously the polymorphic language and its parametricity logic.
(Bernardy et al, 2010–2015; Lasson et al, 2011, 2012.)

Reminder: Pure Type Systems

(See the second lecture of Nov 21st, "Polymorphism all the way up")

Universe: $U \in \mathcal{U}$

Terms, types:

$$
\begin{array}{lll}
A, B, C ::= & x & \text{variables} \\
\mid & \lambda x : A.\, B & \text{abstractions} \\
\mid & A\ B & \text{applications} \\
\mid & \Pi x : A.\, B & \text{dependent function type} \\
\mid & U & \text{universe name}
\end{array}
$$

Notations: $A \to B \stackrel{def}{=} \Pi x : A.\, B$ if x not free in B. $\quad \forall x : A.\, B \stackrel{def}{=} \Pi x : A.\, B$.

A given PTS is obtained by fixing the set $\mathcal{U}$ of universes, and two relations over universes, $\mathcal{A}$ (which universe belongs to which universe?) and $\mathcal{R}$ (which Π-types are well formed?).

Parametricity in a PTS

(Bernardy, Jansson, Paterson, *Proofs for free*, JFP 2012)

Notations:

- For every variable x we assume given two new variables x_1, x_2.
- A_i is the term A where every free variable x is replaced by x_i.

Intuitions:

- Every type $A : U$ becomes a relation $[\![A]\!] : A_1 \to A_2 \to \tilde{U}$ having the general shape of a logical relation.
- Every term $A : B$ becomes a proof that the relation $[\![B]\!]$ relates A_1 with A_2, i.e. $[\![A]\!] : [\![B]\!]\ A_1\ A_2$.
- In particular, when A and B are closed, we get $[\![A]\!] : [\![B]\!]\ A\ A$, which proves the fundamental theorem of logical relations.

(Note: the paper deals with n-place relations. Here, we take $n = 2$.)

Function types

Special case #1: a non-dependent function type, $A \to B$.
The usual condition: two related functions map related arguments to related results.

$$[\![A \to B]\!] = \lambda f_1 : A_1 \to B_1.\ \lambda f_2 : A_2 \to B_2.\ \forall x_1 : A_1.\ \forall x_2 : A_2.\ [\![A]\!]\ x_1\ x_2 \to [\![B]\!]\ (f_1\ x_1)\ (f_2\ x_2)$$

Special case #2: the "for all" from system F, $\forall X : \star.\ B$.
X is interpreted by two types X_1, X_2 and a relation $X : X_1 \to X_2 \to \texttt{Prop}$.

$$[\![\forall X : \star.\ B]\!] = \lambda f_1 : \forall X : \star.\ B_1.\ \lambda f_2 : \forall X : \star.\ B_2.\ \forall X_1 : \star.\ \forall X_2 : \star.\ \forall X : X_1 \to X_2 \to \texttt{Prop}.\ [\![B]\!]\ (f_1\ X_1)\ (f_2\ X_2)$$

Function types

The general case:

$$[\![\Pi x : A.\ B]\!] = \lambda f_1 : \Pi x : A_1.\ B_1.\ \lambda f_2 : \Pi x : A_2.\ B_2.\ \forall x_1 : A_1.\ \forall x_2 : A_2.\ \forall x : [\![A]\!]\ x_1\ x_2.\ [\![B]\!]\ (f_1\ x_1)\ (f_2\ x_2)$$

with, as a consequence:

$$[\![x]\!] = x$$
$$[\![U]\!] = \lambda x_1 : U.\ \lambda x_2 : U.\ x_1 \to x_2 \to \tilde{U}$$

Abstractions, applications, contexts

Consistently with the translation of function types:

$$[\![\lambda x : A.\, B]\!] = \lambda x_1 : A_1.\ \lambda x_2 : A_2.\ \lambda x : [\![A]\!]\ x_1\ x_2.\ [\![B]\!]$$

$$[\![A\ B]\!] = [\![A]\!]\ B_1\ B_2\ [\![B]\!]$$

Every free variable x is translated by 3 variables: two interpretations x_1, x_2 and a relation x. Hence the translation of typing contexts Γ:

$$[\![\emptyset]\!] = \emptyset$$

$$[\![\Gamma, x : A]\!] = [\![\Gamma]\!],\ x_1 : A_1,\ x_2 : A_2,\ x : [\![A]\!]\ x_1\ x_2$$

The fundamental theorem

Modulo technical hypotheses over universes U and their translations $\tilde{U}$:

Theorem

If $\Gamma \vdash A : B$, *then* $[\![\Gamma]\!] \vdash [\![A]\!] : [\![B]\!]\ A_1\ A_2$.

As a corollary, for a closed term: if $\vdash A : B$, then $[\![B]\!]\ A\ A$
(A is related to itself by the logical relation $[\![B]\!]$).

We recover the results for system F by taking $\tilde{\star} = \mathtt{Prop}$:
a type $A : \star$ becomes a relation $[\![A]\!] : A_1 \to A_2 \to \mathtt{Prop}$.

Extensions for free: type constructors (F_ω)

A type constructor such as $\texttt{list} : \star \to \star$ becomes a relation transformer:

$$[\![\star \to \star]\!] = \lambda(F_1, F_2 : \star \to \star).\ \forall(X_1, X_2 : \star).\\ (X_1 \to X_2 \to \tilde{\star}) \to (F_1\ X_1 \to F_2\ X_2 \to \tilde{\star})$$

For instance, $[\![\star \to \star]\!]\ \texttt{list}\ \texttt{list}$ takes a relation $X_1 \to X_2 \to \texttt{Prop}$ between any two types and returns a relation $\texttt{list}(X_1) \to \texttt{list}(X_2) \to \texttt{Prop}$.

Extensions for free: dependent types (LF)

A dependent type such as $\texttt{even} : \texttt{nat} \to \star$ becomes:

$$[\![\texttt{nat} \to \star]\!] = \lambda(F_1, F_2 : \texttt{nat} \to \star).\ \forall(n_1, n_2 : \texttt{nat}).\\ [\![\texttt{nat}]\!]\ n_1\ n_2 \to (F_1\ n_1 \to F_2\ n_2 \to \tilde{\star})$$

Assuming $[\![\texttt{nat}]\!]$ is the identity relation, $[\![\texttt{nat} \to \star]\!]\ \texttt{even}\ \texttt{even}$ takes an integer n and returns a relation $\texttt{even}(n) \to \texttt{even}(n) \to \texttt{Prop}$.

Exercise: what happens to $\texttt{vec} : \star \to \texttt{nat} \to \star$, the constructor of the type $\texttt{vec}\ A\ n$ of lists of A of length n?

Translating the universes

In general we can translate universes identically: $\tilde{U} = U$.

It's the only possible choice for Agda and its hierarchy of universes $\mathtt{Set}_i : \mathtt{Set}_{i+1}$.

In Coq, we have an impredicative $\mathtt{Prop}$ universe in addition to the hierarchy $\mathtt{Set} = \mathtt{Type}_0 : \mathtt{Type}_1 : \cdots : \mathtt{Type}_i : \cdots$.

It is pleasing to take $\widetilde{\mathtt{Prop}} = \widetilde{\mathtt{Set}} = \mathtt{Prop}$ in order to obtain "true" relations ($A_1 \to A_2 \to \mathtt{Prop}$ instead of $A_1 \to A_2 \to \mathtt{Type}$).

But we must take $\widetilde{\mathtt{Type}_i} = \mathtt{Type}_i$ for $i > 0$
(otherwise the typing rule $\vdash \mathtt{Type}_i : \mathtt{Type}_{i+1}$ doesn't translate).

Parametricity is anti-classical

(M. Lasson, 2012)

In other words: classical logic is not parametric.
Indeed, consider the types A corresponding to classical laws:

$\forall X.\, X + (X \to \bot)$	excluded middle
$\forall X.\forall Y.\, ((X \to Y) \to X) \to X$	Peirce's law
$\forall X.\, ((X \to \bot) \to \bot) \to X$	double negation elimination

Assume there exists a term $a : [\![A]\!]$ (a witness for the logical relation for A).
We then get a contradiction.

For instance, for double negation elimination:
we interpret X by any two types and the everywhere-false relation.
$[\![(X \to \bot) \to \bot]\!]\ x_1\ x_2$ is true. $[\![X]\!]\ (f_1\ x_1)\ (f_2\ x_2)$ is false.
Hence $[\![(X \to \bot) \to \bot) \to X]\!]\ f_1\ f_2$ is false.

IV

Advanced topics

Other work on parametricity

We have seen the basic principles of parametricity. It is an active research area, with extensions in many directions. In particular:

General recursion, non-termination:
The theory still applies but "theorems for free" are weaker (because $\bot$ belongs to all types).
(A. M. Pitts, *Parametric Polymorphism and Operational Equivalence*, MSCS 2000.)

Recursive types, mutable state containing functions, etc:
in these cases, the definition of the relations R^t is not well founded by induction over type t. We need other well-foundation principles, such as *step-indexing*.
(Lecture of Jan 9th 2019).

Other work on parametricity

Dynamic typing, gradual typing:

a run-time type test such as `instanceof` in Java can "break" parametricity:

$$(\lambda x.\, \texttt{if}\ x\ \texttt{instanceof}\ \texttt{nat}\ \texttt{then}\ 1\ \texttt{else}\ 0) : \forall X.\, X \to \texttt{nat}$$

Other forms of dynamic typing, such as gradual typing, preserve some of the parametricity properties.

Neis, Dreyer, Rossberg. *Non-parametric parametricity.* JFP 2011.

Ahmed et al. *Theorems for Free for Free: Parametricity, With and Without Types.* ICFP 2017.

Connections with equality and with homotopy:

for closed terms, does $R^t\ (M, N)$ implies $M = N$? Can we prove it? can we postulate it as an axiom? Connections appear with homotopy type theory.

Nuyts, Vezzosi, Devriese, *Parametric Quantifiers for Dependent Type Theory*, ICFP 2017.

Also: the lecture of Jan 23rd 2019.

Other work on parametricity

Parametricity and invariance, in other areas:

Reynolds' parametricity is invariance by change of representation. Notions of parametricity can be seen in other areas where invariance properties play a crucial role, for instance Noether's theorem in physics (existence of a physical quantity that is conserved).

Atkey, *From Parametricity to Conservation Laws, via Noether's Theorem*, POPL 2014.

Connections with symbolic program differentiation:

programs react to small changes in the input data more or less like they react to a change of data representation…

Cai, Giarusso, Rendel, Ostermann, *A theory of changes for higher-order languages – incrementalizing λ-calculi by static differentiation*, PLDI 2014.

V

Further reading

Further reading

Two seminal papers:

- John C. Reynolds, *Types, abstraction and parametric polymorphism*, 1983.
  `http://www.cse.chalmers.se/edu/year/2010/course/DAT140_Types/Reynolds_typesabpara.pdf`
- Phil Wadler, *Theorems for free!*, 1989.
  `http://homepages.inf.ed.ac.uk/wadler/papers/free/free.ps.gz`

Logical relations:

- Syntactic approach: Robert Harper, *Practical Foundations for Programming Languages*, 2016, chapter 48.
- Model-based approach: John C. Mitchell, *Foundations for Programming Languages* , 1996, chapters 5 and 8.

Proofs of strong normalization:

- J.-Y. Girard, Y. Lafont, P. Taylor, *Proofs and Types*, 2003, `http://www.paultaylor.eu/stable/Proofs+Types.html`, chapters 6 and 14.