Free theorems, fast: an overview of parametricity

July 3, 2017

These notes were taken from a talk by Paul Downen during an OPLSS '17 hands-on session. Any errors in transcription or in presentation are mine, not Paul's. Details that I don't understand myself are highlighted in red. I hope you'll be able to furnish the details.

Two ingredients make reasoning about parametricity possible in a programming language. The first ingredient is a mapping from the language's types to a set of terms we consider to have the type. (We'll call this a "semantic interpretation" of a type, although I've just made the term up.) The twist here, though, is to define the semantic interpretation of a type without using typing judgments - no turnstile. The second is a property of this semantic interpretation called *adequacy*, showing (loosely) that terms with a type are in the type's semantic interpretation. So we define semantic interpretations of types for for both STLC and System F, and then show that these semantc interpretations are adequate.

Afterwards, we show how the adequacy theorem for System F can be applied to give us the famous "free theorems", which constrain terms of a type. This reasoning that constrains terms of a type is, again, called parametricity.

1 Semantic interpretations of types in the simply-typed λ calculus

Before we define the semantic interpretation of types to sets of terms in the STLC, we'll first need a utility definition.

Definition 1. The *expansion of a set of terms* $\mathbb{C}$, or $\mathbf{Exp}\,\mathbb{C}$, is the set's closure under reverse reduction. In other words, $N \in \mathbf{Exp}\,\mathbb{C}$ if there exists an $M \in \mathbb{C}$ such that $N \mapsto^* M$.

This is a property we'll want the semantic interpretations of our types to have. Perhaps the real motivation of $\mathbf{Exp}\,\mathbb{C}$, as Dan put it, is to give us more hypotheses to push our proofs through.

Next, we define the semantic interpretations for type themselves. This is an example of a *logical relation*, defined inductively on the sructure of types.

Definition 2. $[\![\_]\!]$ mapping an STLC type to a set of terms.

1. $[\![A \to B]\!] = \{M \in \mathit{Term} \mid \forall N \in [\![A]\!], MN \in [\![B]\!]\}$

2. $[\![A \times B]\!] = \{M \in Term \mid fst\, M \in [\![A]\!], snd\, M \in [\![B]\!]\}$

   The above two definitions are considered "negative", since they are defined in terms of eliminators. Conversely, the following two definitions are considered "positive", since they they are defined in terms of introductory forms.

3. $[\![A + B]\!] = \mathbf{Exp}\{inl\ M \mid M \in [\![A]\!]\} \cup \mathbf{Exp}\{inr\, M \mid M \in [\![B]\!]\}$

4. (For fun.) $[\![\text{Bool}]\!] = \mathbf{Exp}\{\mathbf{true}, \mathbf{false}\}$

Note. A few choices taken by this definition are worth commenting on.

1. Note that we do not define $[\![\_]\!]$ using the typing relation. This becomes more useful in System F, when we want to deal with variables bound by $\forall$, without having to carry around typing contexts.

2. Why does the definition above switch between negative and positive definitions for different types? Observe what happens if we try to define $[\![A+B]\!]$ in positive style, and then use this case in a proof by induction. The eliminator for booleans is given by the typing rule

$$\frac{\Gamma \vdash b : \text{Bool} \quad \Gamma \vdash c : C \quad \Gamma \vdash d : C}{\Gamma \vdash \textbf{if}\, b\, \textbf{then}\, c\, \textbf{else}\, d : C}$$

   The corresponding "positive" definition of $[\![\text{Bool}]\!]$ is then:

$$[\![\text{Bool}]\!] = \{M \in Term \mid \exists\, \text{C a type st.}\ \textbf{if}\ M\ \textbf{then}\ c\ \textbf{else}\ d \in [\![C]\!]\}$$

   Now try to use $[\![\text{Bool}]\!]$ in a proof by induction. In the case where $M \in [\![\text{Bool}]\!]$, the inductive hypothesis would give us information about some type $[\![C]\!]$ that has no connection to $[\![\text{Bool}]\!]$! A similar problem happens with sum types. Put briefly, the eliminators of both sum types and booleans are hypothesised on types that aren't part of the sum or boolean type.

We can now define the adequacy property of the semantic interpretation $[\![\_]\!]$, showing that typed terms are in their type's semantic interpretations. This is a major theorem that we state for its applications, rather than its proof.

Theorem 3. (Adequacy of $[\![\_]\!]$ for STLC.) *If* $\vdash M : A$, *then* $M \in [\![A]\!]$.

For concreteness, here is the adequacy theorem specialised for booleans. This gives us a hint of how adequacy constrains the terms of a type.

Corollary. $\vdash M \in \text{Bool}$ *implies* $M \in \text{Exp}\{\mathbf{true}, \mathbf{false}\}$. *Equivalently,* $\vdash M \in \text{Bool}$ *implies* $M \mapsto^* \mathbf{true}$ *or* $M \mapsto^* \mathbf{false}$.

As example of how the inductive nature of the $[\![\_]\!]$ mapping (and of logical relations in general) help us prove things, we show that semantic interpretations of a type are closed under expansion. We won't need this theorem in the STLC, but it becomes useful when showing adequacy of $[\![\_]\!]$ in System F.

Fact 4. *For all types* A, $[\![A]\!]$ *is closed under expansion with call-by-value evaluation*[1]. *Equivalently,* $[\![A]\!] = \mathbf{Exp}[\![A]\!]$.

[1] This property also holds for call-by-name evaluation. We assume call-by-value evaluation here in order to simplify the proof: given a term M, we must construct an M' and show that $M' \mapsto^* M$. The reduction can be done in one step iif we use call-by-value reduction rules.

Proof. (Partial. We present a single case for each of the positive and negative cases of $[\![\_]\!]$ respectively.) Let A be an arbitrary type. Proceed by induction on A.

Case 1. (A an arrow type $A \to B$.) Let $M \in [\![A \to B]\!]$, and introduce M' st. $M' \to^* M$. We wish to show that $M' \in [\![A \to B]\!]$ - i.e., for all $N \in [\![A]\!]$, we have $M'\,N \in [\![B]\!]$. So let $N \in [\![A]\!]$.

Reduction rules apply to show that

$$\frac{M' \mapsto M}{M'\,N \mapsto M\,N}$$

i.e. that $M\,N \mapsto M'\,N$.

At the same time, since $M \in [\![A \to B]\!]$, we see $M\,N \in [\![B]\!]$.

Finally, since $[\![B]\!]$ is closed under expansion by the inductive hypothesis, both $M\,N \mapsto M'\,N$ and $M\,N \in [\![B]\!]$ together imply $M'\,N \in [\![B]\!]$. This suffices to show that, by the definition of $[\![A \to B]\!]$, $M' \in [\![A \to B]\!]$.

Case 2. (A a sum type $A + B$.) Let $M \in [\![A + B]\!]$. In this case, we wish to show the existence of a term M' st. both $M' \mapsto^* M$, and either

1. There exists an $M'_1 \in [\![A]\!]$ st. $\mathbf{inl}\,M'_1$, or
2. There exists an $M'_2 \in [\![B]\!]$ st. $\mathbf{inl}\,M'_2$.

The assumption that $M \in [\![A + B]\!]$ gives two cases on the form of M.

First, assume that M has the form $\mathbf{inl}\,M_1$, where $M_1 \in [\![A]\!]$. By the induction hypothesis, there exists an $M'_1 \in [\![A]\!]$ with $M'_1 \mapsto^* M_1$. So set $M' = \mathbf{inl}\,M'_1$: we are required to show property 1. holds. We now show that $M' \mapsto M$. Since we have assumed call-by-value evaluation, reduction rules apply to show that

$$\frac{M'_1 \mapsto M_1}{\mathbf{inl}\,M'_1 \mapsto \mathbf{inl}\,M_1}$$

and so $M' \mapsto M$.

Finally, we show that $M' \in [\![A+B]\!]$: since $\mathbf{fst}\,M = M'_1 \in [\![A]\!]$, the conclusion $M' \in [\![A+B]\!]$ follows by the definition of $[\![A + B]\!]$. So we have shown property 1. holds, and the case is done.

The second case requires us to assume that M now has the form $\mathbf{inr}\,M_1$, where $M_1 \in A$. Similar reasoning allows us to give an $M' \mapsto^* M$ satisfying property 2.

□

2 Semantic interpretations of types in System F

We need a few more utility definitions to extend our definition of a type's semantic interpretation to System F requires a few more utility definitions. First, we will have much use for unary predicates on terms that are closed under reduction. We call these *reducibility candidates* (without motivating the name.)

Definition 5. (Reducibility candidates.)

1. A *reducibility candidate* $\mathbf{C}$ is a a set of terms closed under reverse reduction. So reducibility candidates are unary predicates on terms. (Interestingly, note that since $[\![A]\!]$ itself is closed under reverse reduction, $[\![A]\!]$ itself is a reducibility candidate.)

2. $\mathbf{C}\mathbb{R}$ is the set of all reducibility candidates. The term is an abbreviation of Girard's original French.

Second, we will need maps from type variables to reducibility candidates.

Definition 6. We use $\theta[x]$ as the mapping from the type variable x to a reducibility candidate. Further, the notation $\mathbf{C}/x, \theta$ indicates an extension of the map θ with x now mapping to $\mathbf{C}$, a reducibility candidate.

We can now define the semantic interpretation of types in System F, before stating adequacy itself for the semantic interpretation.

Definition 7. $[\![\_]\!]_\theta$ mapping a type to a set of terms, where θ is a map from type variables to reducibility candidates.

1. $[\![A \to B]\!]_\theta$, $[\![A \times B]\!]_\theta$, and $[\![A + B]\!]_\theta$ are defined as for STLC.

2. $[\![\forall X.A]\!] = \{M \,|\, \forall B\, a\, type,\ \forall \mathbf{C} \in \mathbf{C}\mathbb{R},\ M\,[B] \in [\![A]\!]_{\mathbf{C}/X,\theta}\}$. (Here, $M\,[B]$ is the application of the polymorphic type abstraction M to the type B.)

3. (For fun.) $[\![\exists X.A]\!] = \mathbf{Exp}\{(B, M) \,|\, \exists \mathbb{C} \in \mathbf{C}\mathbb{R},\ M \in [\![A]\!]_{\mathbf{C}/X,\theta}\}$

4. $[\![X\, a\, free\, type\, variable]\!] = \theta[X]$, where θ is map defined at X.

We showed in the STLC that, for any type A, the set $[\![A]\!]$ was closed under reverse evaluation - hence, a reducibility candidate. This is also true of $[\![\_]\!]_\theta$ in System F. This property is considerably more useful in System F, since it is used to show adequacy of System F.

Fact 8. *For all types A, $[\![A]\!]_\theta$ is closed under expansion, where θ is an empty mapping from type variables to reducibility candidates. Equivalently, $[\![A]\!]_\theta = \mathbf{Exp}[\![A]\!]_\theta$; equivalently, $[\![A]\!]$ is a reducibility candidate.*

The adequacy property for System F is then as follows:

Theorem 9. (Adequacy of $[\![\_]\!]_\theta$ for System F.) *If $\vdash M : A$, then $M \in [\![A]\!]_\theta$, where θ is an empty mapping from type variables to reducibility candidates.*

3 Free theorems as applications of System F's adequacy

Parametricity is the use of adequacy theorems to constrain the types that can inhabit a term. By stating the adequacy property for a specific type and with specific reducibility candidates, we get "free theorems" on terms. The following fact, for example, uses the adequacy property on terms of types $\forall X.X \to X$.

Theorem 10. If $\vdash M : \forall X.X \to X$, *then* $M =_{\beta\eta} \Lambda X.\lambda(x : X).x$. In other words, there is exactly one term with the type of polymorphic *id*, up to β and η.

Proof. (This proof is quite mechanical: therefore, we italicise non-trivial steps that don't arise from just the definition of adequacy.) Expanding the definitions of adequacy and of $[\![\_]\!]_\theta$ on Λ and $\to$, we get the following: for all types A, reducibility candidates $\mathbf{C}$, and terms x, we have $M\,[A]\,x \in [\![x]\!]_{\mathbf{C}/X,\theta}$. *So set* $\mathbf{C}$ *to* $\mathbf{Exp}\{x\}$. *(This is why we don't define* $[\![\_]\!]_\theta$ *in terms of the typing relation.)* Then, by the definition of $[\![\_]\!]_\theta$ on type variables, $M\,[A]\,x \in \mathbf{Exp}\{x\}$. *This is sufficient to show the desired* β *and* η *equivalence:*

$$\begin{aligned} M &=_\eta \Lambda X.M\,[X] \\ &=_\eta \Lambda X.\lambda(x : X).M\,[X]\,x \\ &=_\beta \Lambda X.\lambda(x : X).x \end{aligned}$$

where the last step follows by the assumption that $\mathrm{M\,[A]\,x} \in \mathbf{Exp}\{\mathrm{x}\}$. $\square$