

Towards a Categorical Understanding of Plotkin-Abadi Logic for Parametric Polymorphism

February 16, 1998

1 Introduction

The idea of parametric polymorphism is that of a single operator that can be used for different data types and whose behaviour is somehow uniform for each type. This concept was first proposed by Strachey [13]. Reynolds [9] uses binary relations to define a uniformity condition for parametric polymorphism. In [1] Plotkin and Abadi proposed a second order logic for second order lambda-calculus; this logic is able to handle parametric polymorphism in the binary relational sense of Reynolds.

We are examining in this paper a categorical framework for this logic. This framework is based on the notion of categorical model of second order lambda-calculus as given, for example, in [8, 12, 10, 5]. Going through the categorical constructions of the model, a slightly unexpected property of quantification over type variables appears. A simple categorical calculation indicates what is the appropriate way to obtain the right adjoint to weakening that models universal quantification. The result coincides with a construction given by Plotkin and Abadi to define the parametricity axiom.

2 Plotkin-Abadi Logic

This section recalls briefly the logic as given in [1]. Types and terms are given by the grammar:

$$\begin{array}{l} \text{Types: } A ::= X \mid A \rightarrow A \mid \forall X.A \\ \text{Terms: } t ::= x \mid \lambda x:A.t \mid u(t) \mid \Lambda X.t \mid t(A) \end{array}$$

where X ranges over type variables (countably many), x over individual variables. Formulae are built up by equations and binary relations, using these terms:

$$\begin{array}{l} \text{Formulae: } \phi ::= (t =_A u) \mid R(t, u) \mid \\ \phi \Rightarrow \psi \mid \phi \wedge \psi \mid \top \mid \perp \mid \\ \forall x:A.\phi \mid \forall X.\phi \mid \forall R \subset A \times B.\phi \end{array}$$

where R ranges over relation variables and $R \subset A \times B$ is a syntactic notation to say that R is a variable for a binary relation between terms of type A and terms of type B . The basic constructs are considered to be implication \Rightarrow and the three sorts of universal quantifiers: over values, over types and over relations between types.

The appropriate environments (contexts) are introduced to give judgements on well-formedness of terms and propositions. Together with the normal axioms and axiom schemes for equality, an axiom schema for parametricity of polymorphism is given. We do not report it as is not going to play a central role in here and requires quite a few more definitions.

3 Some modifications and a model

3.1 From relations to predicates

Starting from the logic in [1], we define one slightly different: instead of (binary) *relation variables* $R \subset A \times B$ and *definable relations*¹ we have predicate variables $P \subset A$ and predicates.

Each formula is a formula-in-context. The contexts we want to consider are threefold: there is a layer (*type context*, γ) which is a list of the type variables we are using. No variable can be repeated twice in a context but type contexts of the same length are identified. Once a type context is fixed, we can form the second and third layer: a *second order context* is formed by adding to γ , a context of individual variables $\Delta \equiv x_1:A_1, \dots, x_k:A_k$ where no individual variable is introduced twice and each type A_i is defined in type context γ ; a *propositional context* is formed in a similar way introducing a finite list of predicate variables $\Pi \equiv P_1 \subset B_1, \dots, P_m \subset B_m$. Finally, we define propositions in contexts $\gamma; \Delta; \Pi$. Well-formed formulae are defined by judgements of the form:

$$\gamma; \Delta; \Pi \vdash \phi \text{ Prop}$$

for example, rules for atomic formulae are:

$$\frac{\gamma; \Delta; \Pi \text{ Context} \quad \gamma; \Delta \vdash t:A \quad P \subset A \text{ in } \Pi}{\gamma; \Delta; \Pi \vdash P(t) \text{ Prop}}$$

$$\frac{\gamma; \Delta; \Pi \text{ Context} \quad \gamma; \Delta \vdash t:A \quad \gamma; \Delta \vdash u:A}{\gamma; \Delta; \Pi \vdash t=_A u \text{ Prop}}$$

Beside the usual rules for connectives, each sort of variable can be quantified over. The rule for universal quantification over type variables is:

$$\frac{\gamma, X; \Delta; \Pi \vdash \phi \text{ Prop}}{\gamma; \Delta; \Pi \vdash \forall X. \phi \text{ Prop}} \quad (X \text{ not free in any type in } \Delta; \Pi)$$

3.2 Introducing the product type

To simplify the structure we need to build we introduce product type $A \times A$. In this way a second order context

$$X_1, \dots, X_n; x_1:A_1, \dots, x_k:A_k$$

is identified with type

$$A_1 \times \dots \times A_k$$

in type context X_1, \dots, X_n .

With this simplification it is possible to model types (or second order contexts) and type contexts in a categorical structure². This is a standard construction ([8, 12, 10, 5]). Here we recall some basic ideas useful to explain the results obtained.

As types are defined in type contexts, they are interpreted as objects of a category **Type** indexed over a category of type contexts **TC**. For each context γ , the category **Type** $_{\gamma}$ is the full subcategory of **Type** whose objects are types in γ . A morphism $A \rightarrow B$ in **Type** $_{\gamma}$ is given by a term $\gamma; x:A \vdash t:B$. The base category **TC** has a distinguished object X , finite products and a terminal object. The distinguished object corresponds to the type context with a single type variable; the terminal object is the empty context; products are given by concatenation of context (renaming

¹a *definable relation* $\rho \subset A \times B$ is obtained from a formula ϕ with free variables $x:A$ and $y:B$ by a kind of abstraction on these variables.

²At the same type one can easily think of translating a binary relation $R \subset A \times B$ into a predicate on the product type.

variables with the same name to avoid repetitions). A morphism $Y_1, \dots, Y_m \xrightarrow{\sigma} X_1, \dots, X_n$ in **TC** is given by the simultaneous substitution on an n -tuple of types:

$$Y_1, \dots, Y_m \vdash X_i \sigma \quad \text{Type}, \quad i = 1, \dots, n$$

for type variables X_1, \dots, X_n . The identity morphism is given by the identical substitution and composition is given by composition of substitutions.

A **Type**-morphism $, ; x:A \rightarrow , ' ; y:B$ is given by a substitution $, \xrightarrow{\sigma} , '$ in **TC** together with a term $, ; x:A \vdash t:B\sigma$.

For any substitution $, ' \xrightarrow{\sigma} ,$ in **Type** there is a functor (*reindexing functor*) $\sigma^*: \mathbf{Type}_\Gamma \rightarrow \mathbf{Type}_{\Gamma'}$ defined by:

$$\begin{aligned} A &\mapsto A\sigma \equiv A[X \mapsto X\sigma] \quad \text{for each } A \text{ in context } \Delta \\ , ; x:A \vdash e:B &\mapsto , ' ; x':A\sigma \vdash e[x \mapsto x']:B\sigma \end{aligned}$$

This indexed structure can also be viewed as a fibration by means of the functor $\mathbf{Type} \rightarrow \mathbf{TC}$ that assign to each type A the type context it is defined on and to each morphism the part that gives the substitution of types for type variables.

The β and η rules for universal quantification establish a natural correspondence:

$$\frac{Y_1, \dots, Y_n, X; x:A \vdash e:B}{Y_1, \dots, Y_n; x:A \vdash e':\forall X.B}$$

that gives right adjoint to the reindexing functor induced by weakening $, ; X \xrightarrow{\sigma} ,$,

Finally, type X in type context X has the property that for each object of **Type** (each type A in type context $,$) there exists a unique morphism $, \xrightarrow{\sigma} X$ in **TC** such that $\sigma^*(X) \cong A$. Such an object is called *generic object*.

4 Predicates in propositional context

The propositions of a logic for simply typed lambda calculus can be modelled in a category indexed over individual variable contexts, in which each fibre is a preordered set, the preorder relation given by the logical consequence relation (cf. [7]). The construction that follows is in the spirit of giving a similar structure for the given second order logic on second order lambda-calculus.

Define **Prop** to have as object predicates $, ; \Delta \vdash \phi \subset A$. A predicate is obtained from a proposition-in-context $, ; x:A; \Pi \vdash \phi(x) \text{ Prop}$ with an abstraction on the free variable x (to be strict we should indicate the predicate obtained from ϕ in a way such as $(x:A).\phi(x)$, as Plotkin and Abadi use for definable relations – we will not be very strict in distinguishing the notation for propositions and predicates, as one can be obtained by the other).

A proposition depends on value variables, propositional variables and (by means of the previous two) type variables. A morphism in **Prop** is given by three “layers” of substitutions, plus a logical consequence. To be precise, if ϕ is a proposition in context $, ' ; \Delta'; \Pi'$ and ψ is a proposition in context $, ; \Delta; \Pi$, a morphism:

$$\phi_{\Gamma'; \Delta'; \Pi'} \rightarrow \psi_{\Gamma; \Delta; \Pi}$$

is given by the following layers:

1. a substitution of types for type variables $, ' \xrightarrow{\sigma} , ;$
2. a substitution of predicates for propositional variables $[P \mapsto P\sigma]$ given by:

$$, ' ; \Pi' \vdash P\sigma \subset A\sigma \quad \text{for any } P \subset A \text{ in } \Pi$$

3. a substitution of terms for individual variables $[y \mapsto e]$ given by:

$$, ' ; \Delta' \vdash e:B\sigma \quad \text{for any } y:B \text{ in } \Delta;$$

4. a logical consequence:

$$\phi \vdash_{\Gamma', \Delta', \Pi'} \psi[X \mapsto X\sigma][P \mapsto P\sigma][y \mapsto e].$$

As base category, consider a category \mathbf{PC} whose objects are propositional contexts $, ; \Pi$. A morphism $, ' ; \Pi' \rightarrow , ; \Pi$ is given by a substitution of types for type variables $, ' \xrightarrow{\sigma} ,$ together with a substitution of predicates for propositional variables (i.e. part 1. and 2. of a morphism in \mathbf{Prop} as given above). The functor $\mathbf{Prop} \rightarrow \mathbf{PC}$ assigns to a predicate the propositional context it is defined on and to a morphism between predicates its first two layers. The following proposition holds:

Proposition 1 $\mathbf{Prop} \rightarrow \mathbf{PC}$ is a model of second order lambda calculus.

Proof: (Sketch) The main two points of the proof are the following:

- The fibres of $\mathbf{Prop} \rightarrow \mathbf{PC}$ are cartesian closed; this is obtained by applying a theorem on fibred adjunction given by C. Hermida in his PhD thesis [4];
- Universal quantification with respect to type variable X is modelled by right adjoint to the reindexing functor induced by weakening $, ; X \rightarrow ,$. Here we need right adjoint to the reindexing functor induced by $\sigma : , , X ; \Pi, P \subset X \rightarrow , ; \Pi$.

Note that $\sigma^* = \theta^* \cdot \rho^*$ where $\rho : , , X ; \Pi, P \subset X \rightarrow , , X ; \Pi$ and $\theta : , , X ; \Pi \rightarrow , , \Pi$.

Let

$$, , X ; \Pi, P \subset X \vdash (x:A).\phi(x)$$

be a predicate in propositional context $, , X ; \Pi, P \subset X$ (we abbreviate $(x:A).\phi(x)$ with ϕ).

One could expect that the image of ϕ under the right adjoint to σ^* is given by \forall -quantification over $P \subset X$ (i. e. right adjoint to θ^*) composed with $\forall X$. What happens instead is that is not possible to apply $\forall X$ to $\forall P \subset X.\phi$. This is because X could be free in the type A of the free variable of ϕ (and indeed, the interesting case is when it *is* free in A)³.

What we would like to obtain from a predicate on type A when we quantify over X is a predicate on type $\forall X.A$. This is obtained by defining:

$$\forall X \forall P \subset X.\phi$$

as the predicate on $\forall X.A$ given by:

$$, ; u:\forall X.A ; \Pi \vdash \forall X \forall P \subset X.\phi[x \mapsto u(X)]$$

It is straightforward to check that this really defines the right adjoint we were looking for. \square

5 Conclusions

To summarise: in the attempt to give an abstract categorical framework for Plotkin and Abadi logic for parametric polymorphism, we build a category of predicates in propositional contexts. This category turns out to be a categorical model of second order lambda-calculus. While the model constructions are verified, an unexpected view of universal quantification over type variables appears. One would expect $\forall X \forall P \subset X.\phi$ (where P is a variable for a predicate on type X) to be the equivalent of $\forall X.A$ (polymorphic type) in the model of second order lambda calculus (i.e. the right adjoint to the reindexing functor induced by weakening with respect to X).

The simple categorical calculation of the needed right adjoint gives the same result as Plotkin and Abadi [1] definition to combine definable relations accordingly to the $\forall X$ constructor obtaining definable relations. This is a basic construction for the definition of the parametricity axiom. This should enable us to give a categorical interpretation of parametricity axiom in the given framework.

³see the side condition on the rule for universal quantification with respect to type variables in Section 3.1.

Acknowledgements

I'd like to thank my supervisor Edmund Robinson for many fruitful discussions and helpful support.

References

- [1] M. Abadi, G. Plotkin. A Logic for Parametric Polymorphism. In M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science **664**, 361–375. Springer-Verlag, March 1993.
- [2] M. Abadi, G. Plotkin. Arities in Relational Parametricity. Unpublished (1993). 135–154 (1985).
- [3] P. J. Freyd, E. Robinson, G. Rosolini. Functorial Parametricity. Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science, 1992, 444–452. Computer Society Press.
- [4] C. A. Hermida, *Fibrations, Logical Predicates and Indeterminates*. PhD Thesis, Edinburgh, 1993.
- [5] B. Jacobs. *Categorical Type Theory*. PhD Thesis, Nijmegen, 1991.
- [6] Q. Ma, J. C. Reynolds. Types, Abstraction and Parametric Polymorphism, part 2. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. A. Schmidt, editors, *Proceeding of the 1991 Mathematical foundations of Programming Semantics Conference*, Lecture Notes in Computer Science **598**, 1–40, Springer-Verlag, Berlin, 1992.
- [7] A. M. Pitts. Notes on Categorical Logic. Unpublished, Cambridge 1989.
- [8] A. M. Pitts. Polymorphism is Set-Theoretic, Constructively. *Category Theory and Computer Science*. D. H. Pitt, A. Poign, D. E. Rydeheard, editors. Lecture Notes in Computer Science **283**, 12–39 (1987).
- [9] J. C. Reynolds. Types, Abstraction and Parametric Polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, 513–523, Amsterdam, 1983. Elsevier Science Publishers B. V. (North-Holland).
- [10] E. P. Robinson. Notes on the Second-Order Lambda Calculus. University of Sussex, Computer Science, Report 4/92, August 1992.
- [11] E. P. Robinson. Logical Relations and Data Abstraction. Unpublished, London 1996.
- [12] R. A. G. Seely. Categorical Semantics for Higher Order Polymorphic Lambda Calculus. *Journal of Symbolic Logic*, **52**, 969–989 (1987).
- [13] C. Strachey. Fundamental Concepts in Programming Languages. *Lecture Notes, International Summer School in Programming Languages*, Copenhagen, Unpublished, August 1967.